Joomla! 1.5 od kuchni. Ponad 130 przepisów!

Autor: James Kennard
Tłumaczenie: Daniel Kaczmarek
ISBN: 978-83-246-2702-8
Tytuł oryginału: Joomla! 1.5 Development Cookbook
Format: B5, stron: 360

Tu znajdziesz rozwiązania najczęściej spotykanych problemów!

- Jak zapewnić możliwość rozwoju rozszerzeń w przyszłości?
- Jak współpracować z bazami danych?
- Jak obsługiwać błędy, wykorzystując mechanizmy Joomla!?

Joomla! to rozbudowany i uniwersalny system zarządzania treścią witryn internetowych, dostępny na prawach open source. Umożliwia tworzenie nie tylko prostych stron internetowych, ale także kompleksowych, rozbudowanych serwisów. Siłą Joomla! jest prostota oraz zaangażowanie twórców w zapewnienie łatwości pracy z tym systemem. Zaawansowani użytkownicy często potrzebują jednak rozwiązań wykraczających poza możliwości dostępnych rozszerzeń. Naprzeciwko tym oczekiwaniom wychodzi elastyczny framework Joomla! – pozwala on programistom dostosowywać się w dowolny sposób i tworzyć własne aplikacje, spełniające wyrafinowane kryteria.

Książka „Joomla! 1.5 od kuchni. Ponad 130 przepisów!" przeznaczona jest dla programistów dysponujących doświadczeniem w implementowaniu rozszerzeń dla tego systemu. Stanowi zbiór ponad 130 prostych, lecz niezwykle użytecznych przepisów, pozwalających rozwiązać praktyczne problemy związane z programowaniem w Joomla!.

Dzięki swemu bogatemu doświadczeniu autor w efektywny i zrozumiały sposób dzieli się posiadaną wiedzą. Przedstawia niewielkie objętościowo przykłady, które ilustrują sposób radzenia sobie z problemami programistycznymi lub projektowymi, powszechnie spotykanymi podczas tworzenia rozszerzeń Joomla!. Profesjonaliści znajdą tu przede wszystkim praktyczne przepisy rozwiązań konkretnych trudności, a początkujący także wiedzę ogólną (związaną chociażby z obsługą błędów w Joomla!), odpowiedzi na pytania oraz sposoby realizacji standardowych zadań. Rozwiązania dotyczą podstawowych zagadnień, czyli m.in. bezpieczeństwa, dostępu do danych, użytkowników, sesji czy możliwości wykorzystania języków narodowych.

- Zapewnienie rozwoju rozszerzeń
- Komunikacja z bazami danych
- Tworzenie źródeł Atom i RSS
- Bezpieczeństwo rozszerzeń
- Obsługa błędów i wyjątków
- Formatowanie stron
- Tworzenie międzynarodowych rozszerzeń
- Komunikacja z użytkownikiem
- Obiekty JObject i tablice
- System plików
- Korzystanie z repozytorium Subversion

Poznaj rozwiązania najczęściej spotykanych w pracy z Joomla! 1.5 problemów, aby tworzyć rozszerzenia lepiej, szybciej i bezpieczniej!

Spis treści

ROZDZIAŁ 3

Praca z bazą danych

Ten rozdział zawiera następujące przepisy:

- Wykonywanie zapytania
- Ładowanie pierwszej komórki ze zbioru wyników zapytania
- Ładowanie pierwszego rekordu z zapytania
- Ładowanie więcej niż jednego rekordu z zapytania
- Obsługa błędów `DBO`
- Tworzenie tabeli `JTable`
- Tworzenie nowego rekordu przy użyciu `JTable`
- Modyfikacja rekordu przy użyciu `JTable`
- Odczytywanie istniejącego rekordu przy użyciu `JTable`
- Usuwanie rekordu przy użyciu `JTable`
- Blokowanie i odblokowywanie rekordu przy użyciu `JTable`
- Zmiana kolejności rekordów przy użyciu `JTable`
- Publikowanie i wycofywanie rekordu z publikacji przy użyciu `JTable`
- Zwiększanie licznika wyświetleń rekordu przy użyciu `JTable`

Wprowadzenie

Większość danych Joomla! jest przechowywanych w bazie danych. Dotyczy to między innymi głównych rozszerzeń, a także rozszerzeń pochodzących od dostawców zewnętrznych. Joomla! często jest określana mianem aplikacji PHP i MySQL. Rzeczywiście, Joomla! korzysta z serwera MySQL, lecz architektura systemu pozwala na użycie również innych serwerów baz danych. Aktualnie wersja 1.5 oficjalnie obsługuje jedynie bazy danych MySQL.

Nazwy wszystkich tabel w bazie danych Joomla! rozpoczynają się od określonego prefiksu. Postać prefiksu jest ustalana globalnie dla całej instalacji systemu, dlatego w odwołaniach do tabel zawsze trzeba używać prefiksu zdefiniowanego dla konkretnej instalacji. Na szczęście prefiksu nie trzeba definiować samodzielnie. Wyobraźmy sobie, że prefiksem jest `jos` i istnieje tabela o nazwie `jos_mojkomponent_foobars`. Zamiennikiem dla prefiksu jest ciąg znaków `#_`, dzięki czemu nazwę tabeli można wyrazić w następujący sposób:

```
#__mojkomponent_foobars
```

Obiektem, którego używa się najczęściej do interakcji z bazą danych, jest globalny obiekt `DBO` (*Database Object*). Jest on uzyskiwany przy wykorzystaniu klasy `JFactory`. Warto zaznaczyć, że do przypisania obiektu zmiennej należy użyć operatora `=&`. Jeżeli operator ten nie będzie użyty, a wersja języka PHP będzie niższa niż 5, utworzona zostanie jedynie kopia obiektu `DBO`.

```
$db =& JFactory::getDBO();
```

Bezpieczeństwo a kod SQL

W trakcie tworzenia zapytań SQL trzeba zachować szczególną ostrożność, ponieważ bardzo łatwo jest narazić się na niebezpieczeństwo. Więcej informacji na temat tworzenia bezpiecznych zapytań SQL znajduje się w przepisach dotyczących języka SQL, w rozdziale 2.

Na potrzeby niniejszego przykładu w każdym przepisie używana będzie tabela zdefiniowana jako tabela 3.1. Oczywiście, nie oznacza to, że w każdym przepisie tabela będzie używana w całości — w odpowiednich przypadkach będziemy bazować wyłącznie na określonych zbiorach danych z tej tabeli.

Oprócz zdefiniowanej tabeli będziemy używać również przykładowych danych, wskazanych w tabeli 3.2.

Aby utworzyć tabelę do celów testowania, najlepiej jest pobrać archiwum przykładowych kodów związanych z tą książką, dostępne na stronie wydawnictwa Helion, pod adresem *ftp://ftp.helion.pl/przyklady/jo15od.zip*.

Przeznaczenie pola `params` nie jest w tym rozdziale wyjaśniane. Pole to służy do rozszerzania bazy danych poza jej pierwotną strukturę. Więcej informacji na ten temat można znaleźć w przepisie dotyczącym obiektów `JParameter` i `JElement`, w rozdziale 9., „Utrzymywanie rozszerzalności i modularności".

Jedną z najbardziej rozbudowanych klas udostępnianych przez Joomla! jest klasa `JTable`. Abstrakcyjna klasa `JTable` umożliwia zaimplementowanie w krótkim czasie interfejsu dla każdej z tabel znajdujących się w bazie danych. Oprócz standardowych elementów, które zwykle wchodzą w skład tego typu klas, `JTable` udostępnia całą gamę metod, za pomocą których bez trudu implementuje się funkcje najczęściej wykonywane w Joomla!, takie jak choćby blokowanie rekordów. Poniższa lista prezentuje wbudowane funkcje udostępniane przez klasę `JTable`:

Tabela 3.1. Definicja tabeli #__mojkomponent_foobars bazy danych na potrzeby przepisów w niniejszym rozdziale

Pole	Typ	NOT NULL	Auto increment	Unsigned	Opis
`id`	`int(11)`	TAK	TAK	TAK	Klucz główny.
`foo`	`varchar(100)`	TAK			Ogólne pole tekstowe, które nie może być puste.
`bar`	`varchar(100)`				Ogólne pole tekstowe, które może być puste.
`checked_out`	`int(11)`	TAK		TAK	Użytkownik, dla którego rekord został zablokowany.
`checked_out_time`	`datetime`	TAK			Czas zablokowania rekordu.
`ordering`	`int(11)`	TAK		TAK	Pozycja, na której powinien znajdować się ten rekord w grupie rekordów.
`published`	`tinyint(1)`	TAK		TAK	Wskazuje, czy rekord jest opublikowany.
`hits`	`int(11)`	TAK		TAK	Liczba wyświetleń rekordu.
`catid`	`int(11)`			TAK	Klucz obcy do tabeli kategorii.
`params`	`text`	TAK			Dodatkowe parametry.

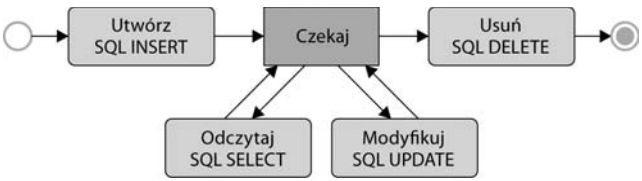
Tabela 3.2. Przykładowe dane dla przepisów z niniejszego rozdziału

id	foo	bar	checked_out	checked_out_time	ordering	published	hits	catid	params
100		NULL	0	0000-00-00 00:00:00	4	1	13	1	
101	Lorem	NULL	0	0000-00-00 00:00:00	3	1	43	1	
102	ipsum	NULL	0	0000-00-00 00:00:00	1	1	72	1	
103	dolor	NULL	62	2009-03-11 11:18:32	2	1	55	1	
104	sit	NULL	0	0000-00-00 00:00:00	1	0	0	2	
105	amet	NULL	0	0000-00-00 00:00:00	2	1	49	2	

- **Wiązanie** — kopiowanie danych z tablicy lub obiektu do obiektu `JTable`.
- **XML** — prezentowanie rekordu w formacie XML.
- **Zarządzanie rekordami** — tworzenie, odczytywanie, modyfikacja i usuwanie rekordów.
- **Weryfikacja poprawności** — sprawdzanie, czy dane w rekordzie odpowiadają zestawowi zdefiniowanych reguł poprawności.
- **Blokowanie** — zapobieganie edycji danego rekordu jednocześnie przez więcej niż jednego użytkownika.
- **Wyznaczanie kolejności** — porządkowanie rekordów zgodnie z preferencjami użytkownika.
- **Publikowanie** — udostępnianie rekordu na widok publiczny lub jego wycofywanie z publikacji.
- **Zliczanie wyświetleń** — rejestrowanie liczby wyświetleń rekordu.

W tym rozdziale wyjaśnimy, jak tworzy się konkretną implementację `JTable`. Ponadto pokazane zostanie, jak korzysta się z poszczególnych funkcji opisanych powyżej.

Zarządzanie rekordami bazuje na paradygmacie **CRUD**, który stanowi skrót od angielskich nazw czynności wykonywanych na rekordach: *Create* (tworzenie), *Read* (odczytywanie), *Update* (modyfikowanie) i *Delete* (usuwanie). Cztery czynności składające się na paradygmat CRUD wyznaczają jednocześnie cykl życia elementu przechowywanego w stałej składnicy danych. Cykl życia elementu wraz ze schematem CRUD przedstawiono na rysunku 3.1. W kontekście `JTable` i CRUD składnicą danych jest baza danych, natomiast elementem jest rekord przechowywany w jednej lub więcej tabel tej bazy albo, mówiąc precyzyjniej, w tabeli reprezentowanej przez dany obiekt klasy `JTable`.



Rysunek 3.1. Paradygmat CRUD i cykl życia rekordu

Czasami może być dość trudno zrozumieć cel klasy `JTable` oraz sposób, w jaki wpasowuje się ona w komponent MVC systemu Joomla!, zwłaszcza jeśli już posiada się model oraz dostęp do bazy danych za pośrednictwem DBO. Aby lepiej zrozumieć kontekst, najlepiej jest myśleć o `JTable` jak o kolejnej warstwie abstrakcji między programistą a bazą danych. Dzięki `JTable` unika się konieczności operowania na nieprzetworzonych danych.

Wykonywanie zapytania

Najbardziej podstawową spośród wszystkich metod klasy `JDatabase` służących do wykonywania zapytania jest metoda `JDatabase::query()`. Metody tej używa się jedynie wówczas, gdy wykonywane zapytanie nie zwraca żadnego zbioru wynikowego, ponieważ metoda zwraca odpowiedzi w postaci nieprzetworzonej. Jeżeli na przykład pomyślnie zostanie wykonane zapytanie `SELECT`, metoda zwróci zasób z danymi wynikowymi. Trudno się jednak spodziewać, by jakikolwiek programista chciał ręcznie operować na zasobie!

Kiedy więc używa się metody `JDatabase::query()`? Mówiąc najprościej, używa się jej wówczas, gdy wynikiem zapytania jest wartość logiczna, czyli gdy wynikiem będzie informacja, czy wykonanie się powiodło, czy nie. Poniżej znajduje się lista rodzajów zapytań na danych, które można wykonywać przy użyciu metody `JDatabase::query()`:

- `DELETE`
- `INSERT`

- RENAME
- REPLACE
- UPDATE

Jak się przygotować?

Aby wykonać zapytanie, trzeba najpierw utworzyć instancję obiektu DBO systemu Joomla!.

```
$db =& JFactory::getDBO();
```

Jak to zrobić?

Pierwszy krok polega na utworzeniu zapytania, które ma zostać wykonane. Poniższy przykład tworzy proste zapytanie DELETE, które usunie wszystkie rekordy z tabeli #__mojkomponent_
➥foobars z wartością ordering większą niż 4:

```
// przygotowanie nazw
$tableName  = $db->nameQuote('#__mojkomponent_foobars');
$columnName = $db->nameQuote('ordering');
// sformułowanie zapytania DELETE
$sql = "DELETE FROM $tableName " .
     . "WHERE $columnName > 4 ";
```

Przed wykonaniem zapytania trzeba wskazać obiektowi DBO, gdzie to zapytanie się znajduje. Brzmi przystępnie i rzeczywiście jest to prosta czynność, ale bardzo często się o niej zapomina:

```
$db->setQuery($sql);
```

Na koniec pozostaje już tylko wykonać zapytanie.

```
if ($db->query()) {
    // zapytanie się powiodło
} else {
    // zapytanie się nie powiodło
}
```

Ponieważ wiemy, że wynikiem zapytania DELETE zawsze będzie wartość true lub false, nic nie stoi na przeszkodzie, by na podstawie wartości zwróconej przez metodę JDatabase::query() ocenić, czy wykonanie zapytania się powiodło, czy nie. Więcej informacji na ten temat można znaleźć w przepisie „Obsługa błędów DBO”, w dalszej części tego rozdziału.

Informacje dodatkowe

Gdy zapytanie zostanie już pomyślnie wykonane, przydatną metodą może się okazać metoda JDatabase::getAffectedRows(). Metoda ta zwraca liczbę rekordów, które były przedmiotem ostatnio wykonywanego zapytania.

```
// zapytanie się powiodło
$affectRowCount = $db->getAffectedRows();
// wyświetlenie potwierdzenia
echo JText::sprintf('USUNIĘTO %u REKORDY(ÓW)', $affectRowCount);
```

Zobacz również

Kolejne trzy przepisy, „Ładowanie pierwszej komórki ze zbioru wyników zapytania", „Ładowanie pierwszego rekordu z zapytania" oraz „Ładowanie więcej niż jednego rekordu z zapytania", prezentują sposoby wykonywania zapytania `SELECT` i pobierania danych zwróconych przez to zapytanie.

Ładowanie pierwszej komórki ze zbioru wyników zapytania

Czasami wykonywane zapytania są bardzo proste i mają na celu odczytanie wyłącznie jednej wartości. Przykładem może być odczytywanie za pomocą funkcji `COUNT()` liczby rekordów, które pasują do zadanych kryteriów, albo sprawdzanie wartości jednej kolumny w rekordzie, którego identyfikator jest dany. W takich przypadkach nie ma potrzeby pobierania całych, złożonych zbiorów danych, aby odczytać interesującą nas wartość. Klasa `JDatabase` udostępnia prosty i szybki sposób odczytywania pierwszej wartości z pierwszego rekordu ze zbioru danych.

Jak się przygotować?

Aby odczytać pojedynczą wartość, należy utworzyć instancję obiektu `DBO` Joomla!.

```
$db =& JFactory::getDBO();
```

Jak to zrobić?

Najpierw trzeba przygotować zapytanie. W poniższym przykładzie za pomocą funkcji agregującej `COUNT()` ustala się liczbę rekordów w tabeli `#__mojkomponent_foobars`. *Jest to modelowa sytuacja, w której odczytywana jest tylko jedna wartość.*

```
// przygotowanie nazw
$tableName = $db->nameQuote('#__mojkomponent_foobars');
// sformułowanie zapytania COUNT
$sql = "SELECT COUNT(*) FROM $tableName";
```

Zanim zapytanie będzie można wykonać, trzeba je wskazać obiektowi `DBO`.

```
$db->setQuery($sql);
```

Na koniec pozostaje wykonać zdefiniowane zapytanie.

```
$total = $db->loadResult();
```

Jeżeli przykładowe zapytanie będzie wykonane na tabeli zdefiniowanej we wprowadzeniu do niniejszego rozdziału, zmiennej `$total` przypisany zostanie wynik zapytania typu `string(1)` o wartości `"6"`. Warto zwrócić uwagę, że choć MySQL zwróci wartość całkowitoliczbową, to będzie ona reprezentowana przez ciąg znaków.

Jak to działa?

W przedstawionym przykładzie zapytanie odczytuje tylko pojedynczą wartość. Co się jednak stanie, jeśli wynikiem zapytania będzie bardziej złożony zbiór danych? Rozważmy zapytanie o następującej treści:

```
SELECT *
FROM `#__mojkomponent_foobars`
WHERE `id` > 103;
```

Wynikiem wykonania zapytania będzie następujący zbiór danych:

104	sit	NULL	0	0000-00-00 00:00:00	1	0	0	2
105	amet	NULL	0	0000-00-00 00:00:00	2	1	49	2

Jeżeli wykonana zostanie metoda `JDatabase::loadResult()`, zwróci wartość z lewego górnego rogu zbioru danych, czyli w tym przypadku wartość `104`.

Informacje dodatkowe

Cóż, nie jest to nic skomplikowanego. Tak naprawdę cały mechanizm działa bardzo łatwo. Jest jednak coś, o czym należy pamiętać. Jako przykładu użyjemy podzbioru danych z tabeli `#__mojkomponent_foobars`, widocznego w tabeli 3.3.

Tabela 3.3. Przykładowy podzbiór danych

id	foo	bar
100		NULL
101	Lorem	NULL
102	ipsum	NULL
103	dolor	NULL
104	sit	NULL
105	amet	NULL

Wykonanie agregującej funkcji `COUNT()` na zbiorze danych z tabeli 3.3 spowoduje, że zwrócona zostanie wartość `6` (w postaci ciągu znaków), co jest jak najbardziej zrozumiałe. Jednak wykonanie ponownie tego samego zapytania, lecz z dodatkową klauzulą `WHERE bar IS NOT NULL`, spowoduje zwrócenie wartości `0` (również będącej ciągiem znaków); ale to również jest jak najbardziej zrozumiałe. Jeżeli zapytamy o wartość `MAX()` z kolumny `id`, otrzymamy wartość `105`. Z kolei jako zawartość kolumny `foo` rekordu o identyfikatorze `100` zwrócony będzie pusty ciąg znaków. Jeżeli będzie wykonane zapytanie o wartość pola `bar` w dowolnym rekordzie, zwrócona zostanie wartość `NULL`.

I co w związku z tym? Przecież wszystko działa idealnie! Jednak jeżeli z jakiegoś powodu wykonanie zapytania się nie powiedzie, również zwrócona zostanie wartość `NULL`. Zależnie od kontekstu zapytania wartość ta może być niejednoznaczna. W przypadku zapytania z funkcją agregującą `COUNT()` łatwo jest zrozumieć wynik `NULL`, ponieważ wiadomo, że prawidłowy wynik powinien być liczbą całkowitą (choć reprezentowaną przez ciąg znaków). Jednak jeżeli zapytanie ma na celu odczytanie wartości z kolumny, w której mogą występować wartości `NULL`, jak ma to miejsce choćby w kolumnie `bar`, wówczas znaczenie zwróconej wartości `NULL` staje się niejasne.

Zobacz również

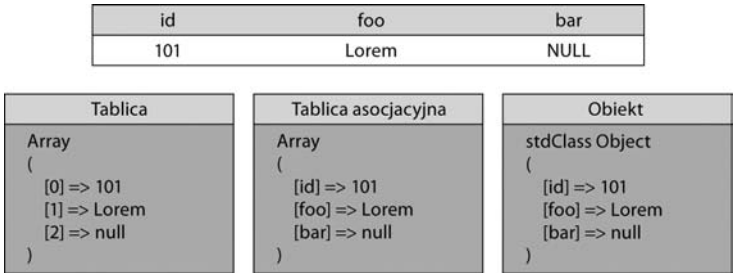
Przepis „Obsługa błędów DBO" opisuje, jak sprawdzać wystąpienia błędów po wykonaniu zapytania.

Ładowanie pierwszego rekordu z zapytania

Dość często zdarza się, że trzeba załadować pierwszy rekord z wyników zapytania. Jeżeli na przykład utworzono komponent, który obsługuje przepisy kulinarne, to gdy użytkownik chce odczytać przepis, wystarczy pozyskać tylko jeden rekord. Łatwo jest ten fakt przeoczyć ze względu na to, że większość programistów jest przyzwyczajona do nawigowania przez zbiory danych, na przykład instrukcją `$record = array_shift($dataset)`. Lecz wykonanie tej samej operacji w Joomla! jest jeszcze łatwiejsze.

Pierwszy rekord można pobrać z zapytania tak naprawdę na trzy sposoby, a wybór konkretnego rozwiązania zależy od formatu, w jakim rekord ma być zwrócony. Dostępne formaty to tablica, tablica asocjacyjna oraz obiekt. Diagram widoczny na rysunku 3.2 ilustruje rekord w postaci takiej, w jakiej występuje w bazie danych. Pod rekordem znajdują się ilustracje trzech dostępnych formatów, w których rekord może zostać zwrócony przy użyciu obiektu klasy `JDatabase`.

W polu `bar` bazy danych znajduje się wartość `NULL`, która jest tożsama z wartością `null` używaną w języku PHP. Nie należy jej jednak mylić z pustym ciągiem znaków, czyli z ciągiem, który nie posiada żadnego znaku. Reprezentacją obiektu jest obiekt klasy `stdClass`. Jest to podstawowa klasa wbudowana w języku PHP, która nie posiada żadnych predefiniowanych składowych.



Rysunek 3.2. Dostępne formaty rekordu zwracanego przez obiekt klasy JDatabase

Najlepiej użyć klasy JTable

Gdy z jednej tabeli trzeba pozyskać tylko jeden rekord i nie są do tego celu używane żadne funkcje języka SQL oraz wiadomo, że mamy do czynienia z wartością klucza głównego, korzystniejsze będzie użycie obiektu klasy `JTable`. Klasa `JTable` udostępnia prosty w użyciu interfejs do tabel znajdujących się w bazie danych. Więcej informacji na ten temat można znaleźć w przepisie „Tworzenie tabeli JTable", w dalszej części rozdziału.

Jak się przygotować?

Aby odczytać pojedynczy rekord, należy utworzyć instancję obiektu `DBO` Joomla!.

```
$db =& JFactory::getDBO();
```

Jak to zrobić?

Najpierw trzeba przygotować zapytanie. Poniższy kod odczytuje z przykładowej tabeli `#__mojkomponent_foobars` rekord o identyfikatorze `101`.

```
// przygotowanie nazw
$tableName = $db->nameQuote('#__mojkomponent_foobars');
$idColumn  = $db->nameQuote('id');
$fooColumn = $db->nameQuote('foo');
$barColumn = $db->nameQuote('bar');
// sformułowanie zapytania COUNT
$sql = "SELECT $idColumn, $fooColumn, $barColumn "
     . "FROM $tableName "
     . "WHERE $idColumn = 101";
```

Przed wykonaniem zapytania trzeba je wskazać obiektowi `DBO`.

```
$db->setQuery($sql);
```

Na końcu pozostaje już tylko wykonać zapytanie. Jak wspomniano już wcześniej, zapytanie można wykonać na trzy sposoby. Przedstawiono je w poniższym przykładzie:

```
// pobranie rekordu w postaci tablicy
$array = $db->loadRow();
// pobranie rekordu w postaci tablicy asocjacyjnej
$associativeArray = $db->loadAssoc();
// pobranie rekordu w postaci obiektu klasy stdClass
$object = $db->loadObject();
```

Jakie jest rzeczywiste działanie każdej z powyższych instrukcji? Odpowiedź znajduje się na diagramie z rysunku 3.2, we wprowadzeniu do tego rozdziału. Kolejne instrukcje zwracają odpowiednio tablicę, tablicę asocjacyjną oraz obiekt i każdy z wyników instrukcji reprezentuje rekord o identyfikatorze `101`.

W zwykłych tablicach numer indeksu zależy od pozycji, dlatego pierwsze pole znajduje się na pozycji `0`. Oznacza to, że aby pozyskać konkretne pole, trzeba najpierw znać jego pozycję w zbiorze danych. Nie jest to wielki problem, lecz cecha ta może stać się źródłem błędów w trakcie utrzymania systemu. Jeżeli na przykład do tabeli będzie dodana nowa kolumna, być może konieczne będzie również zmodyfikowanie znacznej części pozostałego kodu.

Z kolei w tablicach asocjacyjnych i obiektach odwołania do wartości mają postać nazwy pola. Dzięki temu zarówno tablice asocjacyjne, jak i obiekty nie są aż tak wrażliwe na zmiany w strukturze bazy danych, a ich reprezentację łatwo zrozumieć pod względem semantycznym. Dlatego generalnie rzecz biorąc, najlepiej jest używać tablic asocjacyjnych i (lub) obiektów.

Informacje dodatkowe

Ze względów bezpieczeństwa czasami pożądane może być sprawdzenie, czy zapytanie zwróciło tylko jeden wiersz. W niektórych sytuacjach złośliwy użytkownik może zyskać możliwość takiego obejścia zabezpieczeń rozszerzenia, by ładowanych było więcej wierszy niż jeden. Najprostszym przykładem sytuacji, gdy powinno się sprawdzać liczbę wierszy, jest odczytywanie danych z tabeli użytkowników. Pod żadnym pozorem nie powinno się przez przypadek udostępniać takich danych!

Liczbę rekordów odczytanych z bazy danych można sprawdzić metodą `JDatabase::getNumRows()`. Metoda `JDatabase::getNumRows()` zwraca liczbę rekordów, które zostały zwrócone przez ostatnio wykonane zapytanie.

```
if ($db->getNumRows() > 1) {
    // oho, odczytano jakiś ciekawy rekord!
}
```

Uwaga na klauzulę LIMIT, gdy sprawdzana jest liczba wierszy

Metoda `JDatabase::getNumRows()` zwraca liczbę wierszy zwróconych w wyniku wykonania zapytania. Jeżeli zakres zwróconych wierszy zostanie ograniczony przy użyciu klauzuli `LIMIT`, wówczas maksymalna liczba wierszy wynikowych będzie równa wartości klauzuli `LIMIT`. Aby sprawdzić, jaka jest potencjalna liczba wszystkich wierszy wynikowych, należy użyć funkcji agregującej `COUNT()`.

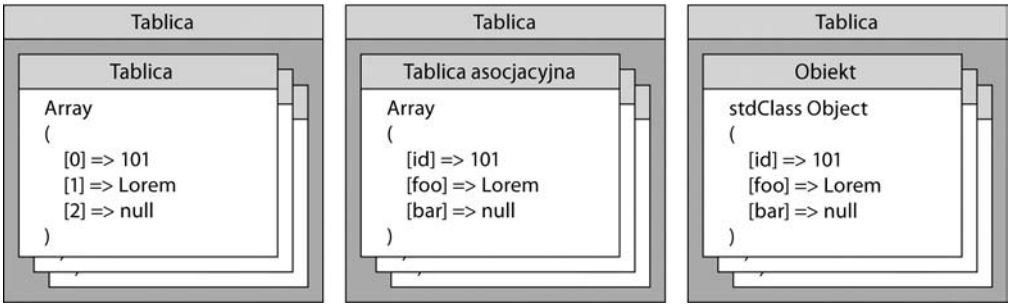
Zobacz również

Przepis „Obsługa błędów DBO" opisuje, jak sprawdzać wystąpienia błędów po wykonaniu zapytania.

Ładowanie więcej niż jednego rekordu z zapytania

Bez względu na to, jaka metoda zostanie wybrana do pozyskania wielu rekordów z bazy danych, zawsze uzyskamy na końcu tablicę rekordów. Inny może być jedynie sposób reprezentacji pojedynczych rekordów w tablicy. Jeżeli używana jest klasa `JDatabase`, wiersz tablicy może mieć jedną z trzech postaci. Może występować jako tablica, tablica asocjacyjna albo obiekt. Diagram widoczny na rysunku 3.3 przedstawia dostępne reprezentacje kilku przykładowych rekordów.

id	foo	bar
101	Lorem	NULL
102	ipsoum	NULL
103	dolor	NULL



Rysunek 3.3. Dostępne reprezentacje rekordów z bazy danych

W polu `bar` bazy danych znajduje się wartość `NULL`, która jest tożsama z wartością `null` używaną w języku PHP. Nie należy jej jednak mylić z pustym ciągiem znaków, czyli z ciągiem, który nie posiada żadnego znaku. Reprezentacją obiektu jest obiekt klas `stdClass`. Jest to podstawowa klasa wbudowana w języku PHP, która nie ma żadnych predefiniowanych składowych.

Jak się przygotować?

Aby uzyskać tablicę rekordów, należy wpierw utworzyć instancję obiektu `DBO` Joomla!.

```
$db =& JFactory::getDBO();
```

Jak to zrobić?

Najpierw trzeba przygotować zapytanie. Poniższy przykładowy kod odczytuje z przykładowej tabeli `#__mojkomponent_foobars` rekordy o identyfikatorach `101`, `102` i `103`.

```
// przygotowanie nazw
$tableName = $db->nameQuote('#__mojkomponent_foobars');
$idColumn  = $db->nameQuote('id');
$fooColumn = $db->nameQuote('foo');
$barColumn = $db->nameQuote('bar');
// sformułowanie zapytania
$sql = "SELECT $idColumn, $fooColumn, $barColumn "
     . "FROM $tableName "
     . "WHERE $idColumn >= 101 AND "
     . "      $idColumn <= 103 ";
```

Przed wykonaniem zapytania trzeba je wskazać obiektowi `DBO`.

```
$db->setQuery($sql);
```

Na końcu pozostaje już tylko wykonać zapytanie. Jak wspomniano już wcześniej, zapytanie można wykonać na trzy sposoby. Przedstawiono je w poniższym przykładzie:

```
// pobranie rekordów w postaci tablicy
$array = $db->loadRowList();
// pobranie rekordów w postaci tablicy asocjacyjnej
$associativeArrays = $db->loadAssocList();
// pobranie rekordu w postaci obiektów klasy stdClass
$objects = $db->loadObjectList();
```

Jakie jest rzeczywiste działanie każdej z powyższych instrukcji? Odpowiedź znajduje się na diagramie z rysunku 3.3, we wprowadzeniu do tego rozdziału. Kolejne instrukcje zwracają odpowiednio tablice, tablice asocjacyjne oraz obiekty i każdy z wyników instrukcji reprezentuje rekordy o identyfikatorach `101`, `102` i `103`.

W zwykłych tablicach numer indeksu zależy od pozycji, dlatego pierwsze pole znajduje się na pozycji `0`. Oznacza to, że aby pozyskać konkretne pole, trzeba najpierw znać jego pozycję w zbiorze danych. Nie jest to wielki problem, lecz cecha ta może stać się źródłem błędów w trakcie utrzymania systemu. Jeżeli na przykład do tabeli będzie dodana nowa kolumna, być może konieczne będzie również zmodyfikowanie znacznej części pozostałego kodu.

Z kolei w tablicach asocjacyjnych i obiektach odwołania do wartości mają postać nazwy pola. Dzięki temu zarówno tablice asocjacyjne, jak i obiekty nie są aż tak wrażliwe na zmiany w strukturze bazy danych, a ich reprezentację łatwo zrozumieć pod względem semantycznym. Dlatego generalnie rzecz biorąc, najlepiej jest używać tablic asocjacyjnych i (lub) obiektów.

Informacje dodatkowe

Tablica, w której zwracane są wyniki zapytania, jest domyślnie zwykłą tablicą, to znaczy tablicą indeksowaną liczbowo, w kolejności zgodnej z kolejnością odczytywania wierszy z bazy danych. Dostępna jest jednak ciekawa opcja, dzięki której można używać indeksów bardziej złożonych. W przypadkach, gdy wiersze posiadają pojedynczą, unikatową wartość, jako indeksu tablicy można użyć właśnie tego klucza. Jeżeli na przykład klucze mają wartości `101`, `102` i `103`, wówczas identyczne wartości mogą mieć klucze tablicy, co widać w przykładowym kodzie:

```
Array
(
    [101] => Array ( [0] => 101 [1] => Lorem [2] => null )
    [102] => Array ( [0] => 102 [1] => ipsum [2] => null )
    [103] => Array ( [0] => 103 [1] => dolor [2] => null )
)
```

Aby uzyskać taki efekt, odpowiednią metodę `JDatabase::load*List()` należy wykonać z opcjonalnym pierwszym parametrem. Parametr przekazany do metody wskazuje jej, która kolumna rekordu reprezentuje klucz. W przypadku metody `JDatabase::loadRowList()` parametr musi być liczbą całkowitą, ponieważ oznacza on indeks kolumny w zbiorze danych. Dla pozostałych dwóch metod wartość parametru musi być ciągiem znaków odpowiadającym nazwie kolumny w zbiorze danych.

```
// pobranie rekordów w postaci tablicy
$arrays = $db->loadRowList(0);
// pobranie rekordów w postaci tablicy asocjacyjnej
$associativeArrays = $db->loadAssocList('id');
// pobranie rekordu w postaci obiektów klasy stdClass
$objects = $db->loadObjectList('id');
```

Zobacz również

Przepis „Obsługa błędów DBO” opisuje, jak sprawdzać wystąpienia błędów po wykonaniu zapytania.

Obsługa błędów DBO

Nie zawsze wszystko idzie zgodnie z planem. Metoda wykonywania zapytania wyznacza jednocześnie sposób, w jaki sprawdza się wystąpienie błędów. Jeżeli na przykład użyto metody `JDatabase::query()`, to w przypadku błędu w wykonaniu zapytania metoda ta zwróci logiczną wartość `false`. Jest to oczywiście w pełni akceptowalna metoda sprawdzania, czy wystąpiły błędy, jednak dostępny jest również służący temu celowi ogólniejszy mechanizm.

Problem ze sprawdzaniem wartości wynikowej jest związany z faktem, że zawsze trzeba wiedzieć, jak każda metoda wykonująca zapytanie sygnalizuje wystąpienie błędu. Kolejnym problemem jest to, że Joomla! może również współpracować z serwerami baz danych innymi niż MySQL, a adaptery dla innych serwerów baz danych mogą inaczej sygnalizować wystąpienie błędu. Na szczęście wystąpienie błędu można rozpoznawać w inny sposób, który bardziej uniezależnia implementowany kod źródłowy od uchwytu `DBO`.

Jak to zrobić?

Metoda `JDatabase::getErrorNum()` zwraca numer błędu wygenerowanego w wyniku wykonania ostatniego zapytania. Jeżeli nie pojawił się żaden błąd, metoda zwraca wartość `0`. Dzięki temu, aby uzyskać informację, czy wystąpił błąd, wystarczy sprawdzić numer błędu.

```
if ($db->getErrorNum() == 0) {
    // żaden błąd nie wystąpił
} else {
    // wystąpiły błędy
}
```

Jak to działa?

Numery błędów zwracane przez metodę `JDatabase::getErrorNum()` są oryginalnymi numerami błędów serwera baz danych. Pewnym problemem jest fakt, że różne serwery baz danych używają różnych kodów błędu. Jeżeli na przykład *wskazana tabela nie istnieje*, serwer MySQL zwróci błąd o kodzie `1146`, zaś dla SQL Servera jest to błąd o kodzie `208`. Z tego powodu metody `JDatabase::getErrorNum()` używa się wyłącznie po to, aby sprawdzić, czy w ogóle wystąpił jakiś błąd.

Informacje dodatkowe

Oprócz kodu błędu można pozyskiwać również treść komunikatu o błędzie. Podobnie jak w przypadku kodów błędów, również treść komunikatów o błędach zależy od używanego serwera baz danych. Wprawdzie obiekt `DBO` jest z technicznego punktu widzenia obiektem klasy `JObject`, lecz do odczytywania ostatnio zwróconego komunikatu o błędzie nie używa się zwykłej metody `JObject::getError()`, ale metody `JDatabase::getError()`.

```
// jeżeli wystąpił błąd
$error = $db->getErrorMsg();
// wyświetlenie komunikatu o błędzie
JError::raiseWarning(500, $error);
```

Istnieje również rozwiązanie alternatywne. Metoda `JDatabase::stderr()` zwraca bardziej rozbudowany komunikat o błędzie.

```
// jeżeli wystąpił błąd
$error = $db->stderr();
// wyświetlenie komunikatu o błędzie
JError::raiseError(500, $error);
```

Wadą odczytywania komunikatów o błędach w taki sposób, jaki przedstawiono powyżej, jest to, że treści tych komunikatów nie są tłumaczone na język bieżący. Generalnie rzecz biorąc, oryginalne treści komunikatów o błędach są używane jedynie wówczas, gdy błąd ma charakter krytyczny i zwracany jest wewnętrzny błąd serwera o kodzie 500, jak w ostatnim przykładzie.

Metoda `JDatabase::stderr()` może zwracać również kod SQL, którego wykonanie spowodowało wygenerowanie błędu. W tym celu metodę wywołuje się z opcjonalnym parametrem `$showSQL` o wartości `true` (domyślnie parametr ten ma wartość `false`). *Nie zaleca się wyświetlania kodu SQL na serwerach działających w środowisku produkcyjnym, ponieważ kod ten zawiera informacje, na podstawie których złośliwy użytkownik może spróbować złamać zabezpieczenia systemu.*

Tworzenie tabeli JTable

Niniejszy przepis prezentuje sposób, w jaki tworzy się klasę `JTable`, która będzie reprezentować przykładową tabelę `#__mojkomponent_foobars`, przedstawioną we wprowadzeniu do tego rozdziału. Na potrzeby przepisu zostaną użyte tylko trzy pierwsze pola tabeli: `id`, `foo` i `bar`.

Jak się przygotować?

Jeżeli klasa `JTable` tworzona jest w ramach komponentu, trzeba najpierw utworzyć folder *tables* (o ile jeszcze nie istnieje). Folder musi się znajdować w głównym folderze administracyjnym rozszerzenia. Na przykład w przypadku komponentu o nazwie `mojkomponent` właściwym folderem będzie folder *administrator/components/com_mojkomponent/tables*.

W (rzadko spotykanym) przypadku, gdy klasa `JTable` tworzona jest dla innego rodzaju rozszerzenia, nie istnieje predefiniowana lokalizacja dla klas `JTable`. Aby wobec konkretnej klasy `JTable`, znajdującej się w lokalizacji alternatywnej, zastosować statyczną metodę `JTable::getInstance()`, należy wskazać klasie `JTable`, w którym folderze dodano podklasy `JTable`. *Warto pamiętać, że istnieje możliwość dodawania więcej ścieżek niż jedna.*

```
JTable::addIncludePath($ścieżkaDoObiektówJTable);
```

Jak to zrobić?

Gdy tworzone są konkretne klasy `JTable`, bardzo ważne są stosowane konwencje nazewnictwa. Plik, w którym definiowana jest klasa, powinien nosić taką samą nazwę jak tabela reprezentowana przez tę klasę (w liczbie pojedynczej, a nie mnogiej). Klasa powinna nosić nazwę zaczynającą się

słowem `Table`, po którym należy umieścić nazwę reprezentowanej tabeli (w liczbie pojedynczej, a nie mnogiej). Na przykład klasa `JTable` dla tabeli `#__mojkomponent_foobars` powinna nosić nazwę `TableFoobar` i znajdować się w pliku *foobar.php*.

W podstawowej implementacji klasy `JTable` pokrywa się zwykle dwie metody: `__construct()` oraz `check()`. Ponadto dla każdego pola tabeli tworzy się zmienne instancji klasy.

> Nie należy dodawać żadnej zmiennej instancji klasy, która nie odnosi się do pola tabeli. Jeżeli konieczne jest zdefiniowanie dodatkowych zmiennych instancji klasy, należy ich nazwy poprzedzić znakiem podkreślenia, co będzie oznaczać, że są one chronione.

Poniższa przykładowa klasa będzie operować na okrojonej wersji tabeli `#__mojkomponent_foobars`.

```
/**
 * Klasa obsługuje tabelę #__mojkomponent_foobars
 */
class TableFoobar extends JTable
{
    /** @var int */
    var $id = null;
    /** @var string */
    var $foo = '';
    /** @var string */
    var $bar = '';
    /**
     * Utworzenie nowej klasy TableFoobar
     */
    function __construct(&$db) {
        parent::__construct('#__mojkomponent_foobars', 'id', $db);
    }
    /**
     * Czy dane są prawidłowe?
     */
    function check() {
        // sprawdzenie poprawności identyfikatora (wartość int albo null)
        if (!preg_match('~^\d+$~', $this->id) || $this->id !== null) {
            $this->setError(JText::_('ID JEST NIEPRAWIDŁOWE'));
            return false;
        }
        // sprawdzenie poprawności pola foo
        if(JString::trim($this->foo) == '') {
            $this->setError(JText::_('TABELA FOOBAR MUSI POSIADAĆ POLE FOO'));
            return false;
        }
        // wszystko w porządku, dane są poprawne!
        return true;
    }
}
```

Gdy gotowa jest już konkretna implementacja klasy `JTable`, można zacząć jej używać. Sposób dostępu do zaimplementowanej klasy zależy od tego, gdzie ma ona zostać wykorzystana. Jeżeli tworzony jest komponent MVC (co jest najczęściej spotykanym przypadkiem), używa się metody `JModel::getTable()`. *Metody* `JModel::getTable()` *zazwyczaj używa się na poziomie modelu. Niemniej jednak należy pamiętać, że jest to metoda publiczna, dzięki czemu można ją stosować również z zewnątrz.*

```
class SomeModel extends JModel {
...
    function someMethod() {
        $table =& $this->getModel('Foobar');
        ...
    }
```

Rozwiązaniem alternatywnym jest bezpośrednie użycie metody `JTable::getInstances()`.

```
$table =& JTable::getInstance('Foobar', 'Table');
```

Warto zwrócić uwagę na sposób, w jaki podano drugi parametr. Jest to prefiks nazwy tabeli, a jego domyślną wartością jest `JTable`. Prefiks domyślny jest używany względem niskopoziomowych implementacji klasy `JTable`, takich jak klasa `JTableUser`.

Jak to działa?

Konstruktor `JTable` przekazuje nazwę tabeli, do której klasa się odnosi, nazwę klucza głównego oraz obiekt `DBO` reprezentujący konstruktor przodka klasy `JTable`. Metoda `check()` pokrywa analogiczną metodę przodka i służy do weryfikacji poprawności danych w zmiennych instancji klasy.

Pokrywanie metody `check()` nie jest obowiązkowe. W rzadkich przypadkach, gdy danych nie obowiązują żadne reguły poprawności, metody `check()` nie trzeba pokrywać. Wprawdzie metoda `check()` służy z założenia do ustalania poprawności danych, ale nic nie stoi na przeszkodzie, by w jej definicji modyfikować również dane, o ile modyfikacje te są stosunkowo nieskomplikowane. Przykładem takiego działania może być skopiowanie wartości do aliasu, jeżeli wartość aliasu nie jest zdefiniowana.

Bardzo istotne jest prawidłowe zrozumienie roli, jaką pełni metoda `check()`. Metodę wykonuje się przed wprowadzeniem jakichkolwiek zmian w tabeli, czyli przed utworzeniem nowego albo zmodyfikowaniem istniejącego rekordu. Rekordy tworzy się i uaktualnia przy użyciu metody `JTable::save()` lub metody `JTable::store()`. Jeżeli wykonywana jest metoda `JTable::save()`, ręczne wywoływanie metody `check()` nie jest już konieczne, ponieważ zostanie ona wywołana automatycznie.

Zobacz również

Następne cztery przepisy opisują, jak tworzy się, odczytuje, zmienia i usuwa rekordy przy użyciu klasy `JTable`.

Tworzenie nowego rekordu przy użyciu JTable

Niniejszy przepis opisuje sposób tworzenia nowego rekordu w bazie danych przy użyciu obiektu klasy `JTable`. W przedstawionym przykładzie nadal używana będzie klasa `JTable`, zdefiniowana w poprzednim przepisie. Dla celów niniejszego przepisu przyjęto założenie, że dane, na podstawie których zostanie utworzony nowy rekord, będą pochodzić z formularza przesłanego metodą `POST`.

Jak się przygotować?

Najpierw trzeba utworzyć obiekt klasy `JTable`. Więcej informacji na temat uzyskiwania instancji klasy `JTable` przedstawiono w poprzednim przepisie.

Jak to zrobić?

Najpierw trzeba pozyskać dane, na podstawie których ma być utworzony nowy rekord. Dane będą powiązane z obiektem `JTable`. Oznacza to, że wartości będą skopiowane z tablicy do tabeli. Nie ma przy tym znaczenia, czy struktura, która zostanie powiązana z obiektem `JTable`, zawiera jakieś dane, ponieważ i tak będą one zignorowane przez obiekt `JTable`. W poniższym przykładzie pobierana jest cała zawartość żądania `POST`. Warto zwrócić uwagę, że ponieważ do pobrania zawartości żądania `POST` używana jest metoda `JRequest::get()`, dane wejściowe pochodzące z tego żądania zostaną od razu zneutralizowane (więcej informacji na ten temat znajduje się w dalszej części tego rozdziału).

```
// wartości, z których ma zostać utworzony nowy rekord
$post = JRequest::get('POST');
```

Musimy się upewnić, że wartość pola `id` (czyli klucza głównego) nie jest zdefiniowana tak, że będzie źródłem konfliktu w momencie tworzenia nowego rekordu. Dlatego polu `id` przypisana zostaje wartość `false`. W ten sposób zyskujemy pewność, że rzeczywiście będzie utworzony nowy rekord, a nie zmieniony rekord już istniejący.

```
// nie podajemy wartości ID
$post['id'] = false;
```

Ostatni krok polega na zapisaniu nowego rekordu przy użyciu metody `JTable::save()`. Metoda `JTable::save()` zwraca wartość logiczną, która wskazuje, czy operacja się udała, czy nie.

```
if (!$table->save($post)) {
    // nie udało się zapisać
}
```

Jeżeli wykonanie metody `JTable::save()` się nie powiedzie, można spróbować użyć metody `JTable::getError()`, aby uzyskać tekstowe informacje o przyczynie błędu. Warto zauważyć, że nie zawsze będą dostępne informacje o błędzie. Metoda `JTable::save()` wywołuje szereg innych metod, w tym metodę `JTable::checkin()`. Jeżeli nie powiedzie się wykonanie metody `JTable::checkin()`, wówczas nie jest definiowany żaden komunikat o błędzie!

Jak to działa?

Metoda `JTable::save()` wykonuje komplet potrzebnych czynności. Mówiąc dokładniej, wywoływane są następujące czynności:

- Zdefiniowanie powiązania ze źródłową tablicą lub obiektem za pomocą metody `JTable::bind()`.
- Weryfikacja poprawności danych przez metodę `JTable::check()`.
- Zapisanie danych metodą `JTable::store()`.
- Zatwierdzenie rekordu metodą `JTable::checkin()`.
- Uporządkowanie rekordów metodą `JTable::reorder()`.

Podobnie jak w przypadku wakacyjnego wyjazdu all inclusive, również nad procesem, który jest wykonywany całościowo, jak przez metodę `JTable::save()`, nie ma się prawie żadnej kontroli. Z tego powodu metoda `JTable::save()` nie zawsze jest najlepszym narzędziem. Analiza kodu źródłowego niektórych komponentów wykaże, że nie zawsze korzystają one z metody `JTable::save()`, a zamiast niej wszystkie potrzebne czynności wykonywane są po kolei. Więcej informacji na ten temat znajduje się w następnym punkcie.

Informacje dodatkowe

Czasami wymagany jest większy zakres kontroli nad danymi, które mają stanowić nowy rekord. Jeżeli na przykład w danych znajduje się pole `text`, w którym można przechowywać kod języka HTML, wówczas użycie zneutralizowanej zmiennej `$post` nie będzie odpowiednim rozwiązaniem, ponieważ wszelkie znaczniki HTML będą usunięte. Aby uwzględnić ten fakt, należy samodzielnie przetworzyć pole tak, aby utrzymane zostały w nim znaczniki HTML.

```
// pole foo może zawierać wartość oryginalną
$post['foo'] = JRequest::getString('foo', '', 'POST',
                                   JREQUEST_ALLOWRAW | JREQUEST_NOTRIM);
```

Przedstawione podejście sprawdza się również w sytuacji, gdy o przetwarzanych wartościach z góry wiadomo, że powinny być określonego typu. Jeżeli na przykład wiadomo, że dana wartość powinna być liczbą całkowitą, można użyć metody `JRequest::getInt()`, aby mieć gwarancję, że uzyskana wartość rzeczywiście jest typu `Integer`. Więcej informacji na temat sposobów korzystania z klasy `JRequest` znajduje się w rozdziale 2., w przepisie „Bezpieczne pobieranie danych z żądania”.

Wiązanie nie zawsze jest potrzebne

Zamiast przeprowadzać wiązanie z tablicą lub obiektem, można ustawić każdy element oddzielnie za pomocą metody `JTable::set()`. Jeżeli używana jest `JTable::set()`, wówczas w wywołaniu metody `JTable::save()` należy przekazać pustą tablicę lub obiekt będący przedmiotem wiązania.

W punkcie „Jak to działa?" powiedziano, że trzecią czynnością, jaką wykonuje metoda `JTable::save()`, jest wykonanie metody `JTable::store()`. To właśnie metoda `JTable::store()` wykonuje najważniejszą czynność, to znaczy wprowadza zmiany w bazie danych. Natomiast problem z metodą `JTable::save()` polega na tym, że nie ma się nad nią prawie żadnej kontroli.

Spójrzmy na przykład. Metoda `JTable::store()` posiada parametr, na podstawie którego można wskazać, czy uaktualniane mają być wartości `null`. Gdy używana jest metoda `JTable::save()`, z góry przyjmuje ona założenie, że wartości `null` nie będą zmieniane. Może to być jednak niepożądane zachowanie, zwłaszcza w przypadku tabeli, której pewne pola mogą zawierać wartości `null`. *Należy pamiętać, że metoda* `JTable::bind()` *nie może ustanawiać wiązania z wartościami* `null`.

Wykorzystanie metody `JTable::reorder()` również jest w pewien sposób ograniczone. Domyślnie zakłada się, że kolejność rekordów w tabeli jest wyznaczana względem pola grupującego i nie może być definiowana jednocześnie w całej tabeli.

Definiowane komunikaty o błędach również nie są specjalnie przydatne. Jeżeli którakolwiek z metod wywoływanych przez `JTable::save()` się nie powiedzie, to nie powiedzie się wykonanie samej `JTable::save()`. Jednak ustalenie, na którym etapie całego procesu pojawił się problem, jest bardzo trudne, a w niektórych przypadkach komunikat z informacją o błędzie w ogóle nie zostanie zdefiniowany!

Poniższy przykład stanowi implementację bardziej kompletnego rozwiązania. Aby łatwiej było je zrozumieć, każdy punkt, w którym obsługiwany jest błąd, oznaczono komentarzem `//błąd`. W takim punkcie proces zostaje przerwany i konieczne jest obsłużenie błędu.

```
// wartości, które mają trafić do nowego rekordu
$post = JRequest::get('POST');
// nie definiujemy ID
$post['id'] = false;
// pole foo może zawierać wartość oryginalną
$post['foo'] = JRequest::getString('foo', '', 'POST', JREQUEST_ALLOWRAW |
➥JREQUEST_NOTRIM);
// powiązanie $post z $table
if (!$table->bind($post)) {
    // błąd
}
// sprawdzenie poprawności danych
if (!$table->check()) {
    // błąd
}
// zapisanie danych w tabeli bazy danych i uaktualnienie wartości null
```

```
if (!$table->store(true)) {
    // błąd
}
// zatwierdzenie rekordu
if (!$table->checkin()) {
    // błąd
}
// uaktualnienie kolejności rekordów w tabeli (bez grupowania)
if (!$table->reorder()) {
    // błąd
}
```

Zobacz również

Poprzedni przepis, „Tworzenie tabeli JTable", pokazuje, jak tworzy się konkretną klasę `JTable`.

Następne dwa przepisy opisują sposób uaktualniania i wczytywania danych przy użyciu danej klasy `JTable`.

Modyfikacja rekordu przy użyciu JTable

Niniejszy przepis opisuje metodę modyfikowania rekordu już istniejącego w bazie danych przy użyciu obiektu klasy `JTable`. Na potrzeby przykładu będzie użyta klasa `JTable`, zaimplementowana w przedostatnim przepisie.

Jak się przygotować?

Najpierw trzeba utworzyć obiekt `JTable`. Sposób tworzenia instancji obiektu `JTable` przedstawiono w przedostatnim przepisie.

Jak to zrobić?

Nietrudno zgadnąć, że modyfikowanie rekordu nie różni się specjalnie od operacji tworzenia rekordu. Tak naprawdę nie różni się prawie niczym oprócz tego, że dodatkowo konieczne jest podanie wartości klucza głównego zmienianego rekordu.

```
// wartości, które mają zostać zapisane w istniejącym rekordzie
// $post zawiera identyfikator ID modyfikowanego rekordu
$post = JRequest::get('POST');
if (!$table->save($post)) {
    // zapisanie danych się nie powiodło
}
```

Ponieważ tworzenie i modyfikowanie rekordu przebiega bardzo podobnie, coraz częściej dąży się do tego, by w ogóle nie traktować obydwóch czynności oddzielnie. W komponencie MVC tworzenie i modyfikowanie rekordu często jest realizowane przez jedną metodę o nazwie `edit()`.

Jak to działa?

Więcej informacji przedstawiono w poprzednim przepisie, w punkcie „Jak to działa?".

Informacje dodatkowe

Więcej informacji przedstawiono w poprzednim przepisie, w punkcie „Informacje dodatkowe".

Odczytywanie istniejącego rekordu przy użyciu JTable

Niniejszy przepis opisuje sposób odczytywania zawartości rekordu już istniejącego w bazie danych przy użyciu obiektu `JTable`. Na potrzeby przykładu użyta będzie klasa `JTable`, zaimplementowana w przepisie „Tworzenie tabeli JTable".

Jak się przygotować?

Najpierw trzeba utworzyć obiekt `JTable`. Sposób tworzenia instancji obiektu `JTable` przedstawiono w przepisie „Tworzenie tabeli JTable".

Jak to zrobić?

Aby wczytać rekord z tabeli, używa się metody `JTable::load()`. Metoda ta ładuje rekord do zmiennych instancji klasy. Pierwszym i jedynym parametrem `JTable::load()` jest wartość klucza głównego rekordu, który ma zostać wczytany. Metoda zwraca wartość logiczną, dzięki czemu możemy od razu sprawdzić, czy wykonanie metody zakończyło się powodzeniem.

```
if ($table->load(JRequest::getInt('id'))) {
    // udało się!
}
```

> Czasami identyfikator rekordu, który ma być wczytany, jest już ustawiony w obiekcie. W takim przypadku do metody `JRequest::load()` nie trzeba przekazywać wartości klucza głównego rekordu.

Wiemy już, jak wczytuje się dane, ale gdzie one trafiają i jak uzyskuje się do nich dostęp? Jak już wiadomo, konkretna implementacja klasy `JTable` zawiera publiczne zmienne instancji, które odnoszą się bezpośrednio do pól w tabeli reprezentowanej przez tę klasę. Dlatego gdy rekord zostanie już załadowany, pochodzące z niego dane można uzyskać metodą `JTable::get()`.

```
$jakieśPole = $table->get('jakieśPole');
```

Usuwanie rekordu przy użyciu JTable

Niniejszy przepis opisuje, jak za pomocą obiektu `JTable` usuwa się rekord istniejący w bazie danych. Na potrzeby przykładu będzie użyta klasa `JTable` zaimplementowana w przepisie „Tworzenie tabeli JTable".

Jak się przygotować?

Najpierw trzeba utworzyć obiekt `JTable`. Sposób tworzenia instancji obiektu `JTable` przedstawiono w przepisie „Tworzenie tabeli JTable".

Jak to zrobić?

W przypadku usuwania danych najważniejsza zasada mówi, że nie należy przywiązywać się emocjonalnie do danych. Naprawdę, przywiązywanie się do danych może być wręcz niezdrowe! A mówiąc poważnie, do usuwania rekordów służy metoda `JTable::delete()`. Jeżeli rekord jest już załadowany, metodę `JTable::delete()` można wywołać bez konieczności podawania jakichkolwiek parametrów — usunie ona wówczas rekord bieżący. Jeżeli natomiast rekord nie został załadowany, do metody `JTable::delete()` można przekazać parametr będący wartością klucza głównego rekordu, który ma być usunięty z bazy.

```
if ($table->delete(JRequest::getInt('id'))) {
    // usunięcie rekordu się powiodło
}
```

Usuwanie rekordu z tabeli, która jest powiązana z innymi tabelami

Za pomocą metody `canDelete()` można sprawdzić, czy istnieją jakiekolwiek zależności, które należy usunąć przed usunięciem samego rekordu. Do metody należy przekazać wartość klucza głównego rekordu, którego zależności trzeba sprawdzić, oraz tablicę definiującą powiązania tabeli, w której ten rekord się znajduje.

Blokowanie i odblokowywanie rekordu przy użyciu JTable

Niniejszy przepis opisuje, jak za pomocą klasy `JTable` ręcznie implementuje się mechanizm blokowania rekordu. Należy pamiętać, że tak zaimplementowany mechanizm jest nadzorowany przez system Joomla!, a nie serwer baz danych, dlatego serwer może unieważnić jego działanie.

Jak się przygotować?

Najpierw trzeba utworzyć obiekt `JTable`. Sposób tworzenia instancji obiektu `JTable` przedstawiono w przepisie „Tworzenie tabeli JTable", we wcześniejszej części tego rozdziału.

Rekordy można blokować jedynie wówczas, gdy tabela, w której rekordy się znajdują, zawiera pola `checked_out` i `checked_out_time`. Pierwsze pole wskazuje użytkownika, który zablokował dany rekord, natomiast drugie pole przechowuje informację o czasie założenia blokady na rekordzie. Typami pól na serwerze MySQL są, odpowiednio, `INT UNSIGNED` oraz `DATETIME`. Nasza przykładowa tabela `#__mojkomponent_foobars` posiada obydwa pola i dzięki temu można w niej blokować rekordy przy użyciu mechanizmu realizowanego przez klasę `JTable`.

Jak to zrobić?

Rekordy blokuje się wówczas, gdy rozpoczyna się edytowanie ich zawartości. Gdy użytkownik na przykład edytuje artykuł w komponencie zarządzania treścią, rekord jest blokowany, aby żaden inny użytkownik nie mógł w tym czasie edytować tego samego artykułu. Przed zablokowaniem rekordu trzeba najpierw sprawdzić, czy rekord nie został już wcześniej przez kogoś zablokowany.

```
// pobranie informacji o bieżącym użytkowniku
$user =& JFactory::getUser();
// załadowanie rekordu
$table->load($id);
// sprawdzenie, czy rekord nie został już wcześniej zablokowany
if ($table->isCheckedOut($user->get('id'))) {
    // ktoś nas uprzedził!
}
```

Jeżeli okaże się, że rekord został już wcześniej przez kogoś zablokowany, standardową czynnością jest przekierowanie przeglądarki do strony, na której będzie zaprezentowana zawartość rekordu i wyświetli się odpowiedni komunikat, informujący, że rekord jest właśnie edytowany przez kogoś innego.

Jeżeli natomiast rekord nie będzie zablokowany, w kolejnym kroku musimy go sami zablokować. Do tego celu służy metoda `JTable::checkout()`.

```
// zablokowanie bieżącego rekordu
$table->checkout($user->get('id'));
```

Gdy edycja rekordu zostanie zakończona, należy rekord odblokować. Czynność tę wykonuje się zwykle wówczas, gdy użytkownik zapisał już wprowadzone zmiany albo zrezygnował z edytowania rekordu. Do odblokowywania rekordu służy metoda `JTable::checkin()`.

```
// odblokowanie bieżącego rekordu
$table->checkin();
```

Informacje dodatkowe

Przykładowy kod, przedstawiony w punkcie „Jak to zrobić", prezentuje standardowy sposób użycia metody `JTable::isCheckedOut()` w odniesieniu do pojedynczego rekordu. W przypadku, gdy nie chcemy ładować rekordu do obiektu `JTable` (co jest przydatne wówczas, kiedy mamy do czynienia z listą elementów, i należy wyświetlić, które z nich zostały zaznaczone, a które nie), metodę `JTable::isCheckedOut()` można wywołać z drugim parametrem, którym będzie wartość pola `checked_out`. Metody tej można używać również statycznie.

```
// pobranie informacji o bieżącym użytkowniku
$user =& JFactory::getUser();
// sprawdzenie, czy rekord nie został już wcześniej zablokowany
if (JTable::isCheckedOut($user->get('id'), $checkedOut)) {
    // rekord jest zablokowany przez innego użytkownika
}
```

> Metoda `JTable::isCheckedOut()` nie tylko wykonuje proste porównywane wartości, ale również sprawdza, czy użytkownik, który zablokował rekord, jest wciąż zalogowany.

Metod `JTable::checkout()` i `JTable::checkin()` można używać także wówczas, gdy rekord, który trzeba zablokować lub odblokować, nie jest aktualnie załadowany. W tym celu odpowiednią metodę należy wywołać z opcjonalnym parametrem `$oid`. Parametr `$oid` wskazuje rekord, który ma być zablokowany lub odblokowany. Aby na przykład zablokować rekord, można wykonać następującą instrukcję:

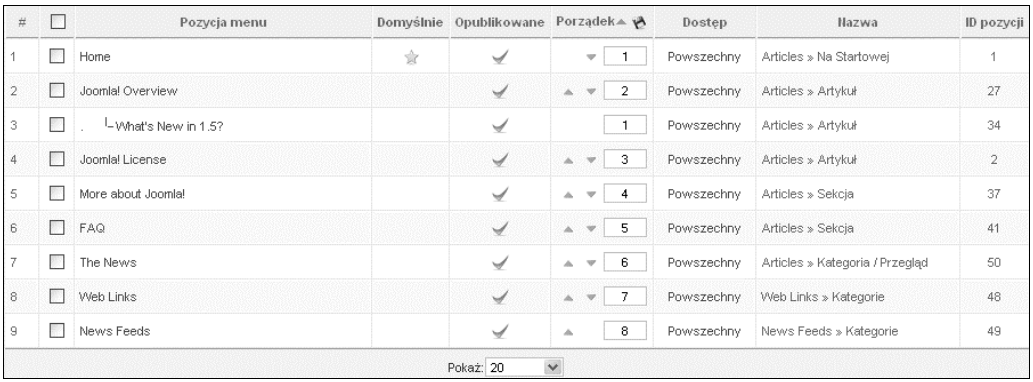
```
// zablokowanie rekordu wskazanego przez $oid
$table->checkout($user->get('id'), $oid);
```

Natomiast do odblokowania rekordu służy następująca instrukcja:

```
// odblokowanie rekordu wskazanego przez $oid
$table->checkin($oid);
```

Zmiana kolejności rekordów przy użyciu JTable

Niniejszy przepis pokazuje, jak za pomocą klasy `JTable` definiuje się kolejność rekordów. Pierwszorzędnym przykładem mogą być menu Joomla!, które administratorzy mogą porządkować w dowolny sposób. Rysunek 3.4 pokazuje, jak określa się kolejność menu w widoku administratora. Warto zwrócić szczególną uwagę na kolumnę *Porządek*.



#		Pozycja menu	Domyślnie	Opublikowane	Porządek	Dostęp	Nazwa	ID pozycji
1		Home			1	Powszechny	Articles » Na Startowej	1
2		Joomla! Overview			2	Powszechny	Articles » Artykuł	27
3		. └–What's New in 1.5?			1	Powszechny	Articles » Artykuł	34
4		Joomla! License			3	Powszechny	Articles » Artykuł	2
5		More about Joomla!			4	Powszechny	Articles » Sekcja	37
6		FAQ			5	Powszechny	Articles » Sekcja	41
7		The News			6	Powszechny	Articles » Kategoria / Przegląd	50
8		Web Links			7	Powszechny	Web Links » Kategorie	48
9		News Feeds			8	Powszechny	News Feeds » Kategorie	49

Pokaż: 20

Rysunek 3.4. Kolumna Porządek wyznacza kolejność menu

Jak się przygotować?

Najpierw trzeba utworzyć obiekt `JTable`. Sposób tworzenia instancji obiektu `JTable` przedstawiono w przepisie „Tworzenie tabeli JTable", we wcześniejszej części tego rozdziału.

Użytkownikom można zezwolić na zmianę kolejności rekordów na podstawie indeksów liczbowych jedynie wówczas, gdy w tabeli znajduje się pole `ordering`. Na serwerze MySQL pole `ordering` jest typu `INT UNSIGNED`, zgodnie zresztą z definicją tabeli `#__mojkomponent_foobars`, przedstawioną we wprowadzeniu do tego rozdziału.

Kolejność rekordów można grupować. Inaczej mówiąc, można wskazywać również inne pola tabeli, aby zdefiniować, do której grupy porządkowej dany rekord należy. W większości przypadków porządkowanie jest wykonywane na podstawie tylko jednego rekordu. W przykładowej tabeli `#__mojkomponent_foobars`, zdefiniowanej we wprowadzeniu do niniejszego rozdziału, grupowanie jest wykonywane względem pola `catid`. Jeżeli odniesiemy się do przykładowych danych przedstawionych we wprowadzeniu, możemy zobaczyć, w jaki sposób grupowanie wpływa na kolejność wartości.

Jak to zrobić?

Do obsługi porządkowania rekordów służą trzy metody klasy `JTable`. Pierwsza z nich to metoda `JTable::getNextOrder()`, która ustala kolejne dostępne miejsce. Zazwyczaj metodę wywołuje się wówczas, gdy nowy rekord dodaje się na końcu listy. Poniższy przykładowy fragment kodu sprawdza kolejne dostępne miejsce przy założeniu, że grupowanie jest wykonywane względem pola `catid` i interesująca nas kategoria jest zdefiniowana przez `$catid`.

```
// przygotowanie grupowania
$db =& JFactory::getDBO();
$group = $db->nameQuote('catid') . ' = ' . intval($catid);
// odczytanie następnego miejsca
$next = $table->getNextOrder($group);
```

Czasami dany sposób porządkowania rekordów staje się niespójny. Na przykład może się zdarzyć, że na liście pojawią się puste miejsca albo konkretne pozycje zostaną wykorzystane więcej niż jeden raz. Aby usunąć wszelkie niespójności, należy wywołać metodę `JTable::reorder()`. Niespójności pojawiają się często po usunięciu jakiegoś rekordu, a czasami także po dodaniu nowego rekordu zamiast wywołania metody `JTable::getNextOrder()` w celu sprawdzenia, na jakiej pozycji rekord powinien się znaleźć.

```
// przygotowanie grupowania
$db =& JFactory::getDBO();
$group = $db->nameQuote('catid') . ' = ' . intval($catid);
// odczytanie następnego miejsca
$next = $table->reorder($group);
```

Jak widać na rysunku 3.4, często się zdarza, że umożliwia się użytkownikowi przesuwanie rekordów w górę i w dół za pomocą zielonych strzałek, dostępnych w kolumnie *Porządek*. Do przesuwania rekordów służy trzecia metoda `JTable::move()`. Przesunięcie rekordu w górę jest symbolizowane przez wartość `-1`, natomiast przesunięcie w dół jest wyrażane jako `+1`.

```
// przygotowanie grupowania
$db =& JFactory::getDBO();
$group = $db->nameQuote('catid') . ' = ' . intval($catid);
// przesunięcie bieżącego rekordu w górę o jedną pozycję
$table->move(-1, $group);
```

Metody `JTable::move()` można użyć w jeszcze jeden, rzadziej spotykany sposób. Jeżeli przesunięcie rekordu zostanie wyrażone wartością `0`, można zmienić pole stanowiące podstawę sortowania bieżącego rekordu i wskazać własną pozycję na posortowanej liście. Nie jest to jednak kompletne rozwiązanie, ponieważ nie przesuwa ono w odpowiedni sposób pozostałych rekordów (o ile takie przesunięcie jest wymagane).

Grupowanie metod sortowania nie zawsze jest potrzebne. Dotyczy to zwłaszcza sytuacji, gdy nie istnieje żaden logiczny element odróżniający rekordy od siebie. W takich przypadkach parametr `$group` może zostać w ogóle pominięty.

Publikowanie i wycofywanie rekordu z publikacji przy użyciu JTable

Niniejszy przepis opisuje sposób, w jaki za pomocą `JTable` publikuje się rekordy i wycofuje się je z publikacji. Podstawowy komponent Joomla! do zarządzania treścią jest najprostszym przykładem narzędzia, w którym za pomocą funkcji publikowania rekordu znajdującego się w tabeli steruje się jego widocznością. Ekran tego komponentu przedstawiono na rysunku 3.5.

#		Tytuł	Opublikowane	Na startowej	Porządek	Dostęp	Sekcja	Kategoria	Autor	Data	Odsłony	ID
1		Home			1	Powszechny			Administrator	19.04.2009	19	46
2		Example Pages and Menu Links			2	Powszechny			Administrator	12.08.2008	45	43
3		What's New in 1.5?			1	Powszechny	About Joomla!	The CMS	Administrator	11.08.2008	93	22
4		Joomla! Overview			2	Powszechny	About Joomla!	The CMS	Administrator	09.08.2008	160	19
5		Extensions			3	Powszechny	About Joomla!	The CMS	Administrator	11.08.2008	106	26

Rysunek 3.5. Publikowanie rekordów w Joomla!

Jak się przygotować?

Najpierw trzeba utworzyć obiekt `JTable`. Sposób tworzenia instancji obiektu `JTable` przedstawiono w przepisie „Tworzenie tabeli JTable", we wcześniejszej części tego rozdziału.

W Joomla! publikowanie to czynność, która polega na ustawieniu flagi, decydującej o tym, czy rekord jest widoczny publicznie, czy nie. *Precyzyjna definicja tego, czy coś jest widoczne publicznie, zależy od tabeli zawierającej dany rekord oraz może także zależeć od zakresu innych uprawnień.* Publikowanie rekordów i wycofywanie rekordów z publikacji jest możliwe wyłącznie wówczas, gdy tabela zawiera pole `published`. Na serwerze MySQL pole `published` jest zdefiniowane jako `TINYINT(1) UNSIGNED`. Wartość 1 w tym polu oznacza, że rekord jest opublikowany, zaś `0` oznacza, że rekord nie jest opublikowany.

Jak to zrobić?

W poniższym przykładowym kodzie następuje opublikowanie grupy rekordów na podstawie wartości parametru `cid` pochodzącej z żądania. W tym przypadku parametr `cid` jest tablicą liczb całkowitych, które wyznaczają identyfikatory jednego lub większej liczby rekordów znajdujących się w przedmiotowej tabeli. Więcej informacji na temat pobierania danych z żądania znajduje się w rozdziale 2., w przepisie „Bezpieczne pobieranie danych z żądania".

```
// pobranie tablicy rekordów, które mają zostać opublikowane
$cids = JRequest::getVar('cid', array(),'REQUEST', 'ARRAY');
// opublikowanie rekordów
$table->publish($cids);
```

Metoda `JTable::publish()` jest inteligentniejsza, niż mogłoby się na początku wydawać. Drugi parametr metody pozwala wskazywać, czy wykonywane jest publikowanie rekordów, czy też rekordy są wycofywane z publikacji. Trzeci parametr z kolei przyjmuje wartość identyfikatora użytkownika, dzięki czemu można oznaczyć rekord jako zablokowany. Blokowanie rekordów jest w tym przypadku bardzo istotne, ponieważ jeżeli struktura tabeli pozwala na blokowanie rekordów i ich odblokowywanie, to nie będzie można dokonywać publikacji ani wycofywać z publikacji tych rekordów, które będą zablokowane. Dzięki podaniu identyfikatora użytkownika w kryteriach wyboru rekordów uwzględniany jest dodatkowo fakt, *czy rekord został zablokowany przez innego użytkownika*.

```
// pobranie tablicy rekordów, które mają zostać opublikowane
$cids = JRequest::getVar('cid', array(),'REQUEST', 'ARRAY');
// odczytanie danych bieżącego użytkownika
$user =& JFactory::getUser();
// wycofanie rekordów z publikacji
$table->publish($cids, 0, $user->get('id'));
```

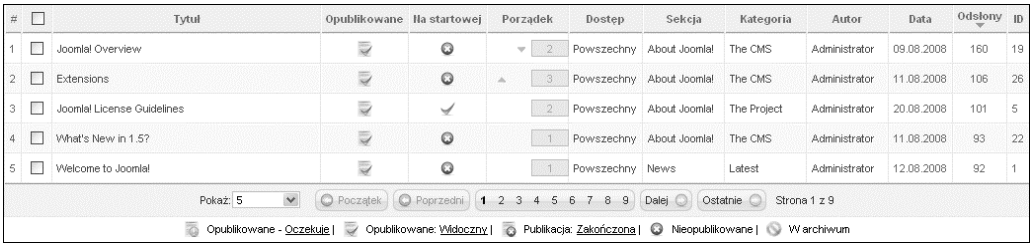
Niespodziewaną, dodatkową czynnością wykonywaną przez metodę `JTable::publish()` jest odblokowywanie rekordów. Jeżeli publikowany jest albo wycofywany z publikacji tylko jeden rekord i tabela posiada pole `checked_out`, to rekord zostanie odblokowany. Jeżeli natomiast czynność jest wykonywana na liczbie rekordów większej niż jeden, wówczas odblokowanie nie będzie mieć miejsca.

Ramy czasowe publikacji

W takich komponentach, jak podstawowy komponent zarządzania treścią, zalecane jest podawanie okresu, przez jaki dany artykuł ma być opublikowany. Klasa `JTable` nie udostępnia jednak żadnego mechanizmu, za pomocą którego można by jawnie podawać okres publikacji artykułu. Stosunkowo łatwo można sobie jednak z tym poradzić. Wystarczy tylko zdefiniować dwa pola typu `DATETIME`, które będą zawierać datę początku i datę końca okresu publikacji (zwykle rekordy te noszą nazwy odpowiednio `publish_up` oraz `publish_down`).

Zwiększanie licznika wyświetleń rekordu przy użyciu JTable

Niniejszy przepis prezentuje sposób użycia klasy `JTable` do zwiększania licznika wyświetleń rekordu. Podstawowy komponent Joomla! do zarządzania treścią jest najprostszym przykładem narzędzia, w którym dzięki licznikowi wyświetleń rekordu można mierzyć jego popularność. Na rysunku 3.6 widać, że najpopularniejszy jest artykuł pod tytułem *Joomla! Overview*, ponieważ został wyświetlony aż **160** razy.



#		Tytuł	Opublikowane	Na startowej	Porządek	Dostęp	Sekcja	Kategoria	Autor	Data	Odsłony	ID
1	☐	Joomla! Overview	✓	✗	2	Powszechny	About Joomla!	The CMS	Administrator	09.08.2008	160	19
2	☐	Extensions	✓	✗	3	Powszechny	About Joomla!	The CMS	Administrator	11.08.2008	106	26
3	☐	Joomla! License Guidelines	✓	✓	2	Powszechny	About Joomla!	The Project	Administrator	20.08.2008	101	5
4	☐	What's New in 1.5?	✓	✗	1	Powszechny	About Joomla!	The CMS	Administrator	11.08.2008	93	22
5	☐	Welcome to Joomla!	✓	✗	1	Powszechny	News	Latest	Administrator	12.08.2008	92	1

Pokaż: 5 | Początek | Poprzedni | 1 2 3 4 5 6 7 8 9 | Dalej | Ostatnie | Strona 1 z 9

Opublikowane - Oczekuje | Opublikowane: Widoczny | Publikacja: Zakończona | Nieopublikowane | W archiwum

Rysunek 3.6. Najpopularniejszy artykuł był wyświetlany 160 razy

Jak się przygotować?

Najpierw trzeba utworzyć obiekt `JTable`. Sposób tworzenia instancji obiektu `JTable` przedstawiono w przepisie „Tworzenie tabeli JTable", we wcześniejszej części tego rozdziału.

Rekordy można publikować i wycofywać z publikacji jedynie wówczas, gdy w tabeli znajduje się pole o nazwie `hits`. Na serwerze MySQL pole `hits` jest definiowane jako pole typu `INT UNSIGNED`. Przykładowa tabela `#__mojkomponent_foobars` umożliwia więc zaimplementowanie mechanizmu publikowania rekordów przy użyciu klasy `JTable`.

Jak to zrobić?

Za każdym razem, gdy użytkownik wyświetla rekord, wystarczy wywołać metodę `JTable::hit()` w następujący sposób:

```
// zwiększenie liczby wyświetleń bieżącego rekordu
$table->hit();
```

Jeżeli rekord, dla którego należy zwiększyć licznik odwiedzin, nie jest aktualnie załadowany, do metody `JTable::hit()` można przekazać wartość klucza głównego tego rekordu.

```
// zwiększenie liczby wyświetleń
$table->hit($id);
```

Metoda `JTable::hit()` zawiera również opcję logowania wykonywanych czynności. Niestety, nie zaimplementowano jeszcze odpowiedniej funkcji, która by z tej opcji korzystała.