```
<?php

require_once( 'CategoryNode.php' );

/**
 * Base Class of Category Traveler
 *
 * @package MediaWiki
 * @author  Zeng Ji(zengji@gmail.com)
 */
abstract class CategoryTravelerBase {

	var $dbr;
	var $sPageTable;
	var $sCategorylinksTable;

	// All category nodes organized by a list. Key is CategoryName, Content is CategoryNode.
	//	This variable is used to quick-orientation while constructing category tree.
	var $categoryList;
	// All category nodes organized by a tree. Key is CategoryName, Content is CategoryNode.
	var $categoryTree;


	function __construct() {
		$this->dbr =& wfGetDB( DB_SLAVE );
		$this->sPageTable = $this->dbr->tableName( 'page' );
		$this->sCategorylinksTable = $this->dbr->tableName( 'categorylinks' );

		$this->categoryList = array();
		$this->categoryTree = array();
	}

	// Allow user to define which category should the traveler start from.
	// If not defined, traveler will start from all level 1 (no parent) categories in datebase.
	function buildCategoryTree($categoryRoot=false) {
		unset($this->categoryList);
		unset($this->categoryTree);
		$this->categoryList = array();
		$this->categoryTree = array();

		$fname = get_class($this) . '::BuildCategoryTree';
		$dbr = $this->dbr;

		// Construct all category nodes, and fill in categoryList.
		$sql = $this->genSQLcategoryAll();
		$res = $dbr->query($sql, $fname);
		while( $obj = $dbr->fetchObject( $res ) ) {
			$cId = $obj->categoryId;
			$cName = $obj->categoryName;
```

```
		$this->categoryList[$cName] = new CategoryNode($cId, $cName);
	}

	if ($categoryRoot == false) {
		// Get level 1 (no parent) categories to travel from.
		$sql = $this->genSQLcategoryTopLevel($this->getNotTopCategoryList());
	} else {
		// Get defined category to travel from.
		$categoryRoot = str_replace( ' ', '_', $categoryRoot);
		$sql = $this->genSQLcategoryByName($categoryRoot);
	}
	$res = $dbr->query($sql, $fname);
	while( $obj = $dbr->fetchObject( $res ) ) {
		$cId = $obj->categoryId;
		$cName = $obj->categoryName;

		// Generate first level categories in tree.
		$categoryNode = $this->categoryList[$cName];
		if (isset($categoryNode)) {
			$this->categoryTree[$cName] = $categoryNode;
		}
		// Recursion to build children tree.
		$this->buildOneCategory(false, $cName);
	}
}

// Recursion Function: Process current category and all its children.
// $cParent: The parent of current processing category node.
//	If current processing category node is top level node, "cParent" should be set to false.
// $cName:   The name of current processing category node.
function buildOneCategory($cParent, $cName) {
	$categoryNode = $this->categoryList[$cName];
	if (isset($categoryNode)) {
		// Avoid the same category node to be travelled more than once.
		// For view of "CategoryTravelerBase", all categories should only has one parent or not.
		//	Although maybe in fact some categories will have more than one parents.
		$categoryNode->hasTravelled = true;

		// Get all children categories, establish children/parent relationship.
		$categoryNode->parent = $cParent;

		$fname = get_class($this) . '::buildOneCategory';
		$dbr = $this->dbr;
		$sql = $this->genSQLcategoryChildren($cName);
		$res = $dbr->query($sql, $fname);

		if ($dbr->numRows( $res ) == 0) {
			// No children categories, recursion ends here.
			return;
		}
```

```
			$categoryNode->children = array();
			while( $obj = $dbr->fetchObject( $res ) ) {

				$childName = $obj->categoryName;
				$childNode = $this->categoryList[$childName];

				if ($childNode->hasTravelled == true)
					continue;
				$categoryNode->children[$childName] = $childNode;
				$this->buildOneCategory($childNode, $childName);
			}
		}
	}

	// Deep recursion to build category tree.
	function travelCategoryTree() {
		$this->travelStart();
		// "0" means "Travel from Root"
		$this->travelOneCategory(0);
		$this->travelEnd();
	}

	function travelOneCategory($level, $categoryNode=false) {
		if ($level==0) {
			// Get TOP Level Categories
			$categoryList = $this->categoryTree;
		} else {
			// Get Next Level Categories
			$categoryList = $categoryNode->children;
		}

		// Process Current Category
		if ($categoryNode != false)
			$this->travel($level, $categoryNode);
		// Process Children of Current Category
		if ($categoryList != false) {
			$categoryCount = count($categoryList);
			$index = 0;
			foreach($categoryList as $cName => $cNode) {
				if ($index == 0)
					$this->travelBeforeFirst($level+1, $cNode);

				$this->travelOneCategory($level+1, $cNode);
				$index = $index + 1;

				if ($index == $categoryCount)
					$this->travelAfterLast($level+1, $cNode);
			}
		}
	}
```

```
abstract function travelStart();
abstract function travelEnd();
// Before travel the first category in one level
abstract function travelBeforeFirst($level, $categoryNode);
abstract function travel($level, $categoryNode);
// After travel the last category in one level
abstract function travelAfterLast($level, $categoryNode);

///////////  Utility Function  ////////////////

// Generate SQL: Retrieve category information by name
function genSQLcategoryByName($cName) {
	$sql = "SELECT page_id AS categoryId, page_title AS categoryName ";
	$sql.= "FROM $this->sPageTable ";
	$sql.= "WHERE page_namespace=" . NS_CATEGORY . " AND page_title=\"" . $cName . "\"";


	return $sql;
}

// Generate SQL: Retrieve children categories by name
function genSQLcategoryChildren($cName) {
	$sql = "SELECT cl.cl_from AS categoryId, page_title AS categoryName ";
	$sql.= "FROM $this->sPageTable INNER JOIN $this->sCategorylinksTable AS cl ";
	$sql.= "ON page_id=cl.cl_from AND (page_namespace=" . NS_CATEGORY . ") ";
	$sql.= "AND (cl.cl_to='" . $cName . "') ";
	$sql.= "ORDER BY page_title";

	return $sql;
}

// Generate SQL: Retrieve all "non-TOP level" categories.
//	"non-TOP level" means has parents.
function genSQLcategoryNotTopLevel() {
	$sql = "SELECT cl.cl_from AS categoryId, page_title AS categoryName ";
	$sql.= "FROM $this->sPageTable INNER JOIN $this->sCategorylinksTable AS cl ";
	$sql.= "ON page_id=cl.cl_from AND (page_namespace=" . NS_CATEGORY . ") ";
	$sql.= "ORDER BY page_title";

	return $sql;
}

// Generate SQL: Retrieve all categories.
function genSQLcategoryAll() {
	$sql = "SELECT page_id AS categoryId, page_title AS categoryName ";
	$sql.= "FROM $this->sPageTable ";
	$sql.= "WHERE page_namespace=" . NS_CATEGORY . " ";
	$sql.= "ORDER BY page_title";
```

```
		return $sql;
	}

	// Generate SQL: Retrieve all "TOP level" categories.
	//	Use "NOT IN" clause in SQL query. "excludeList" should be all "non-TOP level" categories' ID.
	function genSQLcategoryTopLevel($excludeList) {
		$sql = "SELECT page_id AS categoryId, page_title AS categoryName ";
		$sql.= "FROM $this->sPageTable ";
		if ($excludeList) {
			$sql.= "WHERE page_namespace=" . NS_CATEGORY . " AND (page_id NOT IN (" .
$excludeList . "))";
		} else {
			$sql.= "WHERE page_namespace=" . NS_CATEGORY;
		}

		return $sql;
	}

	// Return a string which contains all "non-TOP level" categories' ID.
	function getNotTopCategoryList() {
		$fname = get_class($this) . '::getNotTopCategoryList';
		$sql = $this->genSQLcategoryNotTopLevel();
		$dbr = $this->dbr;
		$res = $dbr->query($sql, $fname);

		// Query result is blank.
		if ($dbr->numRows( $res ) == 0) {
			$dbr->freeResult( $res );
			return false;
		}

		// Generate a string of categories' ID list.
		//	For example: 1234,3721,4567
		$ret = '';
		while( $obj = $dbr->fetchObject( $res ) ) {
			if( isset( $obj->categoryId ) ) {
				$ret .= $obj->categoryId . ',';
			}
		}
		$ret = substr($ret, 0, strlen($ret)-1);   // Delete tail comma
		$dbr->freeResult( $res );

		return $ret;
	}
}

?>
```