MultiDB - PDO

In questa sezione sono descritte le configurazioni necessarie per interfacciarsi con i diversi driver per la connessione con i DBMS.

MySQL

Attualmente non prevista la gestione.

PostgreSQL

La dll del driver è già presente in php. Nel file '**php.ini**' occorre abilitare la seguente istruzione:

```
extension=php_pdo_pgsql.dll
```

Connection.ini Postgres

```
[CITYWARE]
dbms=pgsql
host=[host]
database=[database]
realname=cityware
user=[user]
pwd=[pwd]
fieldskeycase=UPPER
drivertype=PDO
charset=LATIN9
stripspaces=true
defaultString=blank
searchSequenceField=naming
```

MS SQL Server (da 2008 in avanti)

Installazione driver

Scaricare il driver direttamente dal sito di Microsoft utilizzando questo link. Prendere SQLSRV32.EXE eseguirlo e copiare su '...\xampp\php\ext' solo i file php_pdo_sqlsrv_56_ts.dll e php_sqlsrv_56_ts.dll generati dall'exe.

ATTENZIONE! Dalla versione 3.1 in avanti, è necessario scaricare anche il driver ODBC per MS Sql Server, sempre dal sito di Microsoft, utilizzando questo link.

Configurazione php.ini

Dopo aver installato i driver, nel file '**php.ini**' occorre aggiungere:

```
extension=php_sqlsrv_56_ts.dll
extension=php_pdo_sqlsrv_56_ts.dll
```

Oracle

Installazione driver

Utilizzare Instant Client (a seconda della versione di Oracle)

Documentazione di riferimento

php.net

Guida all'installazione su CentOS

Oracle PDO e CentOS

Configurazione php.ini

Dopo aver installato i driver, nel file '**php.ini**' occorre abilitare la seguente istruzione:

```
extension=php_pdo_oci.dll
```

Connection.ini Oracle

```
[CITYWARE]
dbms=oracle
host=[nome tnsnames.ora]
sid=[SID]
database=[database]
realname=cityware
user=[user]
pwd =[pwd]
drivertype=PDO
dateFormat=YYYY-MM-DD
defaultString=blank
searchSequenceField=naming
```

Connection.ini Oracle FAILOVER:ON

```
[CITYWARE]
dbms=oracle
host=X:1521 Y:1521 ;;host separate dello spazio
hostProperties=FAILOVER:ON  ;;proprietà separate dallo spazio: value (es:
LOAD_BALANCE:ON FAILOVER:ON)
servicename= CITYWARE_UNIONE
realname=cityware
user=cityware
pwd=cityware
drivertype=PDO
dateFormat=YYYY-MM-DD
stripspaces=1
defaultString=blank
searchSequenceField=naming
```

Connection.ini mssqlServer

```
[CITYWARE]
dbms=mssqlserver
host=[host\istanza] oppure [host:port]
database=[database]
realname=cityware
drivertype=PDO
user=[user]
pwd=[pwd]
dateFormat=YMD
stripspaces=true
defaultString=blank
searchSequenceField=naming
```

Struttura delle classi in itaEngine

config.ini

Per abilitare PDO, modificare la seguente impostazione nel file config.ini:

```
[dbms]
dbengine=DBPDO
attivaTransazioni=1
```

lib/DBPDO

- **ItaDB**: Classe Proxy
- **ItaDBFacade**: Classe Facade per interfaccia “legacy” con i driver
- **PDPFacade**: Classe Facade per interfaccia PDO con i driver
- **ItaDBError**: Errori specifici DB
- **PDODriver**: Superclasse Driver DB
- **PDOHelper**: Helper funzioni PDO
- **PDOTableDef**: Classe che contiene la definizione della tabella
- **PDOTableDefFactory**: Factory per creazione TableDef
- **PDOTableDefValidator**: Classe per validazione tabella
- **PDO<Driver specifico>**: Classi specifiche per i driver (MS SQL Server, Oracle, Postgres…)

Cache TableDef

Le definizioni delle tabelle vengono messe in cache. Per attivare la cache, occorre modificare il file config.ini aggiungendo le seguenti righe: (Oltre alle definizioni delle colonne, la tableDef contiene anche le eventuali relazioni)

```
[cache]
type=FILE
root=C:/Works/PhpDev/dati/itaTest/cache
```

Validatori

Per la validazione, la logica è la seguente:

- Se esiste il validatore specifico per un model, questo viene utilizzato
- Se non esiste il validatore specifico per un model, viene effettuata la fallback sulla classe 'PDOTableDefValidator.class'

Per stabilire il validatore specifico, la regola è la seguente:

- Nome model = cwbBtaNazion.php
- Classe validator specifica = validators/cwbBtaNazionValidator.php

Può essere necessario escludere alcuni campi dalla validazione, implementando il metodo **initExcludeFields**, es:

```
public function initExcludeFields() {
    parent::initExcludeFields();

    $this->addExcludeField("ISO3166_A2");
    $this->addExcludeField("ISO3166_A3");
    $this->addExcludeField("ISO3166_N3");
    $this->addExcludeField("CODNAZICO");
    $this->addExcludeField("CODNAZIMC");
```

```
    $this->addExcludeField("CODGOVE");
    $this->addExcludeField("LLPIVANAZ");
    $this->addExcludeField("CODAGEO");
    $this->addExcludeField("CODEST_770");
}
```

Gestione relazioni

E' possibile effettuare il salvataggio e cancellazione (in transazione) dei dati aggiuntivi rispetto al record principale. I tipi di relazione gestiti sono:

- One-To-One
- One-To-Many
- Many-To-One

Occorre effettuare l'override del metodo '**caricaDatiAggiuntivi**'. Come esempio, vedere 'cwbBtaGrunaz_Rel'.

Transazioni

Per ogni connessione, esiste una sola transazione (non vengono annidate).

Abilitare sul Hook.ini il listner:

```
[connectionPerRequestPDOHook.php]
active=1
```

Gestione delle connessione e della transazione

La connessione viene salvata come resources all'interno del App→$utente in modo da avere una connessione per request Punti fermi:

1. Elenco numeratoAd ogni request la connessione al db muore
2. Il nome della connessione deve essere univico (ci saranno la connessione 'cityware', 'numeratori',ecc)
3. le query e il service utilizzano sempre la stessa connessione per request presa dal App::$utente

Transazioni: Punti fermi:

1. Esiste un unica transazione aperta per ogni connessione (logica multidb)
2. Service: di default effettua begin_Transaction (se non la trova aperta) e commit_transaction
3. Service: è possibile non fare la commit\rollback se viene impostato l'attributo '$startedTransaction'(o in fase di builder o impostando un “set”)
4. E' possibile gestire manualmente la transazione usando 'manual' in fase di addTransaction.
5. Nel aprire manualmente una nuova sessione c'è il controllo delle transazioni aperte. In caso sia aperta lancia eccezione.
6. Microsoft Mssql con driver installato su macchine linux non è permesso gestire le transazioni.

La chiusura (commit\rollback) deve essere fatta esplicitamente con l'attributo 'manual'. Qualsiasi altro salvataggio\cancellazione\inserimento lanciato con attivo lo stato manual non interferisce con la transazione.

Lettura e scrittura dei binari con il PDO

Per leggere e scrivere correttamente un “blob” effettuare le seguenti operazioni:

1. Nella function che ritorna la stringa sql (esempio cwbLibDB_BGE function getSqlLeggiBgeAgidConfEfil) è obbligatorio specificare tutti i campi in maniera di selezione.
2. Sempre nella stessa function per i campi binari usare questa sintassi per formattare il campo binario in selezione '$this→getCitywareDB()→adapterBlob(“NOMECAMPOBINARIO”)'.Questo risolve il problema del fetch null con oracle su piattaforma linux.
3. Sulla leggi effettiva (esempio cwbLibDB_BGE function leggiBgeAgidConfEfil..) passare un array con oggetto e metodo. E'la callback chiamata per il database mssql per caricare il binario sull'array principale.
4. Implementare questa callback seguendo l' esempio cwbLibDB_BGE leggiBgeAgidConfEfilBinary.

Gestione degli ordinamenti con il PDO in fase di paginazione

In fase paginazione è obbligatorio per i database Mssql\Oracle arrivare a paginare i dati con un ordinamento impostato. L'ordinamento può essere impostato:

- Sul generator
- Sulla “lib” che effettua la query controllando sempre il flag “ $excludeOrderBy” per gestire l'ordinamento dalla datatable (intestazione colonne)
- Nel caso che non venga passato nessun ordinamento di default viene utilizzare la chiave primaria del modello solo se 'noCrud' == false