|  |
| --- |
| GIBMIT |
| KNW - Stored Procedures |
| Modul 128 |
|  |
| **Benjamin Jenni** |
| **14.05.2017** |

|  |
| --- |
|  |

Inhalt

[Namenskonvention 2](#_Toc482564313)

[Definitionen 2](#_Toc482564314)

[Datenbank-Migration 3](#_Toc482564315)

[Kurzbeschreibung 3](#_Toc482564316)

[Ein-/ Ausgabeparameter 3](#_Toc482564317)

[Aufrufbeispiele 3](#_Toc482564318)

[Code 3](#_Toc482564319)

[Orte auslagern 8](#_Toc482564320)

[Kurzbeschreibung 8](#_Toc482564321)

[Ein-/ Ausgabeparameter 8](#_Toc482564322)

[Aufrufbeispiele 8](#_Toc482564323)

[Code 8](#_Toc482564324)

[Benutzer erstellen 11](#_Toc482564325)

[Kurzbeschreibung 11](#_Toc482564326)

[Ein-/ Ausgabeparameter 11](#_Toc482564327)

[Aufrufbeispiele 11](#_Toc482564328)

[Code 11](#_Toc482564329)

[Lernende archivieren 14](#_Toc482564330)

[Kurzbeschreibung 14](#_Toc482564331)

[Ein-/ Ausgabeparameter 14](#_Toc482564332)

[Aufrufbeispiele 14](#_Toc482564333)

[Code 14](#_Toc482564334)

# Namenskonvention

Um die Datenbank lesbarer und einheitlicher zu gestalten, wird bei der Namensgebung überall das gleiche Schema angewendet.

## Definitionen

1. Alle Attributnamen und Tabellennamen beginnen werden klein geschrieben.
2. Primary Keys werden nach folgendem Muster benannt: „id\_name“.
3. Foreign Keys werden nach folgendem Muster benannt: „fk\_name“.
4. Stored Procedures werden nach folgendem Muster benannt: „sp\_name“.
5. Cursor werden nach folgendem Muster benannt: „cr\_name“.

# Datenbank-Migration

## Kurzbeschreibung

Diese Stored Procedure nimmt die veraltete Datenbank mit ihren wirren Bezeichnungen und falschen Typen und wandelt sie in eine neue Datenbank um, in welcher die Namen der Namenskonvention entsprechen. Die Daten werden zudem aus der alten Datenbank in die neue übernommen.

## Ein-/ Ausgabeparameter

Parameter werden keine benötigt.

## Aufrufbeispiele

Untenstehend findet sich eine Variante, wie man die Stored Procedure aufrufen kann.

CALL sp\_migration();

## Code

**sp\_migration.sql**

DROP PROCEDURE IF EXISTS `sp\_migration`;

CREATE PROCEDURE `sp\_migration`()

BEGIN

DROP DATABASE IF EXISTS `schoolinfo\_neu`;

/\* Define settings \*/

SET DEFAULT\_STORAGE\_ENGINE = InnoDB;

SET CHARACTER\_SET\_CLIENT = utf8;

SET CHARACTER\_SET\_RESULTS = utf8;

SET CHARACTER\_SET\_CONNECTION = utf8;

SET COLLATION\_SERVER = utf8\_unicode\_ci;

SET COLLATION\_DATABASE = utf8\_unicode\_ci;

SET COLLATION\_CONNECTION = utf8\_unicode\_ci;

SET SQL\_MODE = 'ALLOW\_INVALID\_DATES';

CREATE DATABASE `schoolinfo\_neu`;

/\* ----- Create tables ------ \*/

/\* Create table for classes \*/

CREATE TABLE `schoolinfo\_neu`.`klassen` (

`id\_klasse` INT(10) NOT NULL AUTO\_INCREMENT,

`id\_lehrer` INT(10) DEFAULT NULL,

`name` VARCHAR(50) NOT NULL,

`beschreibung` VARCHAR(255) NOT NULL,

PRIMARY KEY (`id\_klasse`),

INDEX `id\_lehrer` (`id\_lehrer`)

);

/\* Create table for companies \*/

CREATE TABLE `schoolinfo\_neu`.`lehrbetriebe` (

`id\_lehrbetrieb` INT(10) NOT NULL AUTO\_INCREMENT,

`name` VARCHAR(100) NOT NULL,

`strasse` VARCHAR(100) DEFAULT NULL,

`nummer` VARCHAR(10) DEFAULT NULL,

`plz` VARCHAR(10) DEFAULT NULL,

`ort` VARCHAR(50) DEFAULT NULL,

`kanton` VARCHAR(50) DEFAULT NULL,

`land` VARCHAR(50) DEFAULT NULL,

PRIMARY KEY (`id\_lehrbetrieb`)

);

/\* Create table for orientations \*/

CREATE TABLE `schoolinfo\_neu`.`fachrichtungen` (

`id\_fachrichtung` INT(10) NOT NULL AUTO\_INCREMENT,

`name` VARCHAR(50) NOT NULL,

PRIMARY KEY (`id\_fachrichtung`)

);

/\* Create table for students \*/

CREATE TABLE `schoolinfo\_neu`.`lernende` (

`id\_lernender` INT(10) NOT NULL AUTO\_INCREMENT,

`anrede` VARCHAR(25) NOT NULL,

`name` VARCHAR(50) NOT NULL,

`vorname` VARCHAR(50) NOT NULL,

`geschlecht` ENUM ('M', 'F') NOT NULL,

`fk\_klasse` INT(10) DEFAULT NULL,

`bm` BOOL NOT NULL,

`fk\_fachrichtung` INT(10) NOT NULL,

`fk\_lehrbetrieb` INT(10) DEFAULT NULL,

`strasse` VARCHAR(50) DEFAULT NULL,

`plz` VARCHAR(10) DEFAULT NULL,

`ort` VARCHAR(50) DEFAULT NULL,

PRIMARY KEY (`id\_lernender`),

FOREIGN KEY (`fk\_klasse`) REFERENCES `klassen` (`id\_klasse`),

INDEX `fk\_klasse` (`fk\_klasse`),

FOREIGN KEY (`fk\_fachrichtung`) REFERENCES `fachrichtungen` (`id\_fachrichtung`),

INDEX `fk\_fachrichtung` (`fk\_fachrichtung`),

FOREIGN KEY (`fk\_lehrbetrieb`) REFERENCES `lehrbetriebe` (`id\_lehrbetrieb`),

INDEX `fk\_lehrbetrieb` (`fk\_lehrbetrieb`)

);

/\* Create table for modules \*/

CREATE TABLE `schoolinfo\_neu`.`module` (

`id\_modul` INT(10) NOT NULL AUTO\_INCREMENT,

`name` VARCHAR(50) NOT NULL,

`beschreibung` VARCHAR(255) DEFAULT NULL,

PRIMARY KEY (`id\_modul`)

);

/\* Create table for marks \*/

CREATE TABLE `schoolinfo\_neu`.`noten` (

`fk\_lernender` INT(10) NOT NULL,

`fk\_modul` INT(10) NOT NULL,

`note\_knw` DECIMAL(3, 2) NOT NULL,

`note\_erfahrung` DECIMAL(3, 2) NOT NULL,

`datum\_erfahrungsnote` DATETIME DEFAULT NULL,

`datum\_knw` DATETIME DEFAULT NULL,

FOREIGN KEY (`fk\_lernender`) REFERENCES lernende (`id\_lernender`)

ON DELETE CASCADE,

INDEX `fk\_lernender` (`fk\_lernender`),

FOREIGN KEY (`fk\_modul`) REFERENCES module (`id\_modul`),

INDEX `fk\_modul` (`fk\_modul`)

);

/\* ----- Create column for old PKs ----- \*/

ALTER TABLE `schoolinfo\_neu`.`klassen`

ADD COLUMN id\_old INT(10) NOT NULL;

ALTER TABLE `schoolinfo\_neu`.`fachrichtungen`

ADD COLUMN id\_old INT(10) NOT NULL;

ALTER TABLE `schoolinfo\_neu`.`lehrbetriebe`

ADD COLUMN id\_old INT(10) NOT NULL;

ALTER TABLE `schoolinfo\_neu`.`lernende`

ADD COLUMN id\_old INT(10) NOT NULL;

ALTER TABLE `schoolinfo\_neu`.`module`

ADD COLUMN id\_old INT(10) NOT NULL;

ALTER TABLE `schoolinfo\_neu`.`noten`

ADD COLUMN id\_old\_lernender INT(10) NOT NULL,

ADD COLUMN id\_old\_modul INT(10) NOT NULL;

/\* ----- Copy Data ----- \*/

/\* Migrate orientations \*/

INSERT INTO `schoolinfo\_neu`.`fachrichtungen` (`id\_old`, `name`)

SELECT

`idrichtung` AS `id\_old`,

`richtung` AS `name`

FROM `schoolinfo1282017`.`richtung`;

INSERT INTO `schoolinfo\_neu`.`klassen` (id\_old, id\_lehrer, name, beschreibung)

SELECT

`idklasse` AS `id\_old`,

`klassenlehrer` AS `id\_lehrer`,

`name` AS `name`,

`realname` AS `beschreibung`

FROM `schoolinfo1282017`.`klasse`;

/\* Migrate companies \*/

INSERT INTO `schoolinfo\_neu`.`lehrbetriebe` (`id\_old`, `name`, `strasse`, `nummer`, `plz`, `ort`, `kanton`, `land`)

SELECT

`id\_Lehrbetrieb` AS `id\_old`,

`FName` AS `name`,

`FStrasse` AS `strasse`,

`FHausNr` AS `nummer`,

`FPlz` AS `plz`,

`FOrt` AS `ort`,

`FKanton` AS `kanton`,

IF(`FLand` = 'Schweiz', 'CH', `FLand`) AS `land`

FROM `schoolinfo1282017`.`lehrbetriebe`;

UPDATE `schoolinfo\_neu`.`lehrbetriebe`

SET

`strasse` = IF(`strasse` = '', NULL, `strasse`),

`nummer` = IF(`nummer` = '', NULL, `nummer`),

`kanton` = IF(`kanton` = '', NULL, `kanton`),

`land` = IF(`land` = '', NULL, `land`);

/\* Migrate classes \*/

INSERT INTO `schoolinfo\_neu`.`klassen` (`id\_old`, `id\_lehrer`, `name`, `beschreibung`)

SELECT DISTINCT

CONCAT('foobar-', `lernender`.`klasse`) AS `name`,

CONCAT('foobar-', `lernender`.`klasse`) AS `beschreibung`,

NULL AS `id\_lehrer`,

`lernender`.`klasse` AS `id\_old`

FROM `schoolinfo1282017`.`lernende` AS `lernender`

INNER JOIN `schoolinfo1282017`.`klasse` AS `k`

ON `k`.`idklasse` = `lernender`.`klasse`

WHERE `k`.`idklasse` IS NULL;

/\* Migrate students \*/

INSERT INTO `schoolinfo\_neu`.`lernende` (`id\_old`, `anrede`, `name`, `vorname`, `geschlecht`, `fk\_klasse`, `bm`, `fk\_fachrichtung`, `fk\_lehrbetrieb`, `strasse`, `plz`, `ort`)

SELECT

`lernender`.`Lern\_id` AS `id\_old`,

`lernender`.`anrede` AS `anrede`,

`lernender`.`name` AS `name`,

`lernender`.`vorname` AS `vorname`,

IF(`lernender`.`geschlecht` REGEXP '[Mm].\*', 'M', 'F') AS `geschlecht`,

`klassen`.`id\_klasse` AS `fk\_klasse`,

IF(`lernender`.`bm` = 0, FALSE, TRUE) AS `bm`,

`fachrichtungen`.`id\_fachrichtung` AS `fk\_fachrichtung`,

`lehrbetriebe`.`id\_Lehrbetrieb` AS `fk\_lehrbetrieb`,

IF(`lernender`.`strasse` = '', NULL, `lernender`.`strasse`) AS `strasse`,

IF(`lernender`.`plz` = '', NULL, `lernender`.`plz`) AS `plz`,

`lernender`.`ort` AS `ort`

FROM `schoolinfo1282017`.`lernende` AS `lernender`

LEFT JOIN `schoolinfo\_neu`.`klassen` ON `klassen`.`id\_old` = `lernender`.`klasse`

LEFT JOIN `schoolinfo\_neu`.`fachrichtungen` ON `fachrichtungen`.`id\_old` = `lernender`.`richtung`

LEFT JOIN `schoolinfo\_neu`.`lehrbetriebe` ON `lehrbetriebe`.`id\_old` = `lernender`.`lehrbetrieb`;

/\* Migrate modules \*/

INSERT INTO `schoolinfo\_neu`.`module` (`id\_old`, `name`, `beschreibung`)

SELECT

`idmodul` AS `id\_old`,

`m\_name` AS `name`,

`modulname` AS `beschreibung`

FROM `schoolinfo1282017`.`modul`;

/\* Migrate marks \*/

INSERT INTO `schoolinfo\_neu`.`noten` (`fk\_lernender`, `fk\_modul`, `note\_erfahrung`, `note\_knw`, `datum\_erfahrungsnote`, `datum\_knw`, `id\_old\_lernender`, `id\_old\_modul`)

SELECT

`lernender`.`id\_lernender` AS `fk\_lernender`,

`modul`.`id\_modul` AS `fk\_modul`,

`note`.`erfahrungsnote` AS `note\_erfahrung`,

`note`.`knw\_note` AS `note\_knw`,

`note`.`dat\_erfa` AS `datum\_erfahrungsnote`,

`note`.`dat\_knw` AS `datum\_knw`,

`lernender`.`id\_old` AS `id\_old\_lernender`,

`modul`.`id\_old` AS `id\_old\_modul`

FROM `schoolinfo1282017`.`noten` AS `note`

LEFT JOIN `schoolinfo\_neu`.`module` AS `modul` ON `modul`.`id\_old` = `note`.`module\_idmodule`

LEFT JOIN `schoolinfo\_neu`.`lernende` AS `lernender` ON `lernender`.`id\_old` = `note`.`lernende\_idLernende`

WHERE `lernender`.`id\_lernender` IS NOT NULL;

/\* ----- Delete old IDs ----- \*/

ALTER TABLE `schoolinfo\_neu`.`klassen`

DROP COLUMN `id\_old`;

ALTER TABLE `schoolinfo\_neu`.`fachrichtungen`

DROP COLUMN `id\_old`;

ALTER TABLE `schoolinfo\_neu`.`lehrbetriebe`

DROP COLUMN `id\_old`;

ALTER TABLE `schoolinfo\_neu`.`lernende`

DROP COLUMN `id\_old`;

ALTER TABLE `schoolinfo\_neu`.`module`

DROP COLUMN `id\_old`;

ALTER TABLE `schoolinfo\_neu`.`noten`

DROP COLUMN `id\_old\_lernender`,

DROP COLUMN `id\_old\_modul`;

END;

# Orte auslagern

## Kurzbeschreibung

Um die Datenbank zu normalisieren müssen unter anderem die Ortschaften ausgelagert werden. Dafür wird eine neue Tabelle erstellt, worauf dann die Lernenden und die Lehrbetriebe mit einem Fremdschlüssel verweisen.

## Ein-/ Ausgabeparameter

Ein- und Ausgabeparameter werden nicht benötigt, beziehungsweise sind nicht vorhanden.

## Aufrufbeispiele

Folgende Codezeile zeigt, wie man die Stored Procedure aufrufen kann.

CALL sp\_places();

## Code

**sp\_places.sql**

DROP PROCEDURE IF EXISTS `sp\_places`;

CREATE PROCEDURE `sp\_places`()

BEGIN

/\* Declare variables \*/

DECLARE id\_ort INT(10);

DECLARE name VARCHAR(50);

DECLARE plz VARCHAR(10);

DECLARE lernende\_fertig BOOLEAN DEFAULT FALSE;

DECLARE lehrbetriebe\_fertig BOOLEAN DEFAULT FALSE;

DECLARE cr\_lernende CURSOR FOR SELECT

`id\_lernender`,

`plz`,

`ort`

FROM `schoolinfo\_neu`.`lernende`;

DECLARE cr\_lehrbetriebe CURSOR FOR SELECT

`id\_lehrbetrieb`,

`plz`,

`ort`

FROM `schoolinfo\_neu`.`lehrbetriebe`;

DECLARE CONTINUE HANDLER FOR NOT FOUND SET lernende\_fertig = TRUE;

DROP TABLE IF EXISTS `schoolinfo\_neu`.`ortschaften`;

/\* Create table \*/

CREATE TABLE `schoolinfo\_neu`.`ortschaften` (

`id\_ort` INT(10) AUTO\_INCREMENT,

`name` VARCHAR(50),

`plz` VARCHAR(10),

PRIMARY KEY (`id\_ort`),

UNIQUE KEY `name\_plz` (

`name`,

`plz`

)

);

/\* Add foreign key columns \*/

ALTER TABLE `schoolinfo\_neu`.`lernende`

ADD COLUMN `fk\_ort` INT(10) DEFAULT NULL,

ADD FOREIGN KEY (`fk\_ort`) REFERENCES `schoolinfo\_neu`.`ortschaften`(`id\_ort`);

ALTER TABLE `schoolinfo\_neu`.`lehrbetriebe`

ADD COLUMN `fk\_ort` INT(10) DEFAULT NULL,

ADD FOREIGN KEY (`fk\_ort`) REFERENCES `schoolinfo\_neu`.`ortschaften`(`id\_ort`);

OPEN cr\_lehrbetriebe;

/\* Loop to fetch data \*/

myloop: LOOP

IF lehrbetriebe\_fertig

THEN

BEGIN

FETCH cr\_lernende

INTO id\_ort,

name,

plz;

END;

ELSE

BEGIN

FETCH cr\_lehrbetriebe

INTO id\_ort,

name,

plz;

END;

END IF;

IF lernende\_fertig

THEN

IF lehrbetriebe\_fertig

THEN

BEGIN

CLOSE cr\_lernende;

LEAVE myloop;

END;

ELSE

BEGIN

SET lehrbetriebe\_fertig = TRUE;

CLOSE cr\_lehrbetriebe;

OPEN cr\_lernende;

SET lernende\_fertig = TRUE;

ITERATE myloop;

END;

END IF;

END IF;

SET @lernende\_fertig = lernende\_fertig;

IF plz IS NOT NULL

THEN

SET @id\_ort = NULL;

SELECT

`id\_ort`,

`name`,

`plz`

INTO

@id\_ort,

@name,

@plz

FROM `schoolinfo\_neu`.`ortschaften`

WHERE `name` = name AND `plz` = plz

LIMIT 1;

IF @id\_ort IS NULL

THEN

BEGIN

INSERT INTO `schoolinfo\_neu`.`ortschaften` (

`name`,

`plz`

) VALUES (

name,

plz

);

SET @id\_ort = LAST\_INSERT\_ID();

END;

END IF;

IF lehrbetriebe\_fertig

THEN

UPDATE `schoolinfo\_neu`.`lernende`

SET `id\_ort` = @id\_ort

WHERE `id\_lernender` = id\_ort

LIMIT 1;

ELSE

UPDATE `schoolinfo\_neu`.`lehrbetriebe`

SET `id\_ort` = @id\_ort

WHERE `id\_lehrbetrieb` = id\_ort

LIMIT 1;

END IF;

SET lernende\_fertig = @lernende\_fertig;

END IF;

END LOOP;

/\* Drop old columns \*/

ALTER TABLE `schoolinfo\_neu`.`lernende`

DROP COLUMN `plz`,

DROP COLUMN `ort`;

/\* Drop old columns \*/

ALTER TABLE `schoolinfo\_neu`.`lehrbetriebe`

DROP COLUMN `plz`,

DROP COLUMN `ort`;

END;

# Benutzer erstellen

## Kurzbeschreibung

Es sollen neue Benutzer erstellt werden können. Mit dieser Stored Procedure werden Werte verarbeitet und abgespeichert.

## Ein-/ Ausgabeparameter

|  |  |  |
| --- | --- | --- |
| Parameter | Datentyp | Beschreibung |
| benutzer | VARCHAR(255) | Name des neuen Benutzers |
| passwort | VARCHAR(255) | Passwort des neuen Benutzers |
| hostname | VARCHAR(255) | Hostname |
| berechtigung | VARCHAR(255) | Die Berechtigungen, die vergeben werden sollen |
| zugriffsort | VARCHAR(255) | Die Datenbank, Tabelle oder Attribute, für welche die Berechtigungen gelten sollen. |

## Aufrufbeispiele

CALL sp\_users("user","123456aA","127.0.0.1","READ,WRITE","schoolinfo\_neu");

## Code

**sp\_users.sql**

DROP PROCEDURE IF EXISTS `sp\_users`;

CREATE PROCEDURE `sp\_users`(

IN benutzer VARCHAR(255),

IN passwort VARCHAR(255),

IN hostname VARCHAR(255),

IN berechtigung VARCHAR(255),

IN zugriffsort VARCHAR(255)

)

BEGIN

/\* Input Validation \*/

IF benutzer IS NULL OR benutzer = ''

THEN

SIGNAL SQLSTATE 'ERROR'

SET MESSAGE\_TEXT = 'Ungültige Eingabe: Benutzer darf nicht leer sein!';

END IF;

IF passwort IS NULL OR passwort = ''

THEN

SIGNAL SQLSTATE 'ERROR'

SET MESSAGE\_TEXT = 'Ungültige Eingabe: Passwort darf nicht leer sein!';

END IF;

IF hostname IS NULL OR hostname = ''

THEN

SIGNAL SQLSTATE 'ERROR'

SET MESSAGE\_TEXT = 'Ungültige Eingabe: Hostname darf nicht leer sein!';

END IF;

IF berechtigung IS NULL OR berechtigung = ''

THEN

SIGNAL SQLSTATE 'ERROR'

SET MESSAGE\_TEXT = 'Ungültige Eingabe: Berechtigungsfeld darf nicht leer sein!';

ELSE

SET @berechtigungen = UPPER(berechtigung);

SET @concattedstring = CONCAT(berechtigung, ',');

berechtiungsloop: LOOP

SET @berechtigung = SUBSTRING\_INDEX(@berechtigungen, ',', 1);

SET @concattedstring = TRIM(SUBSTRING(@concattedstring, LENGTH(@berechtigung + 2)));

SET @berechtigung = TRIM(@berechtigung);

IF @berechtigung NOT REGEXP '\\s\*(\\w|\\s)\*\\s\*'

THEN

BEGIN

SET @nachricht = CONCAT('Ungültige Berechtigung: ', @berechtigung);

END;

END IF;

IF @concattedstring IS NULL OR @concattedstring = ''

THEN

LEAVE berechtiungsloop;

END IF;

END LOOP;

END IF;

IF zugriffsort IS NOT NULL OR zugriffsort <> ''

THEN

BEGIN

IF zugriffsort REGEXP '\\A(\\w+|\\\*)\\z'

THEN

SET @art = 'DB';

IF zugriffsort REGEXP '%\*%'

THEN

SET @zugriffsort = CONCAT(zugriffsort, '.\*');

ELSE

SET @zugriffsort = zugriffsort;

END IF;

ELSEIF zugriffsort REGEXP '\\A(\\w+|\\\*)\\.(\\w+|\\\*)\\z'

THEN

SET @zugriffsort = zugriffsort;

SET @art = 'TBL';

ELSEIF zugriffsort REGEXP '\\A(\\w+|\\\*)\\.(\\w+|\\\*)\\.(\\w+|\\\*)\\z'

THEN

SET @zugriffsort = SUBSTRING\_INDEX(zugriffsort, '.', 2);

SET @eigenschaft = TRIM(SUBSTRING(zugriffsort, LENGTH(@zugriffsort) + 2));

SET @art = 'COL';

SET @concattedstring = CONCAT(@berechtigungen, ',');

SET @berechtigungen = '';

berechtigungsloopneu: LOOP

SET @berechtigung = SUBSTRING\_INDEX(@concattedstring, ',', 1);

SET @concattedstring = TRIM(SUBSTRING(@concattedstring, LENGTH(@berechtigung) + 2));

SET @berechtigungen = CONCAT(@berechtigungen, TRIM(@berechtigung), ' (', @eigenschaft, '), ');

IF @concattedstring IS NULL OR @concattedstring = ''

THEN

LEAVE berechtigungsloopneu;

END IF;

END LOOP;

SET @berechtigungen = SUBSTRING(@berechtigungen, 1, LENGTH(@berechtigungen) - 2);

ELSE

SIGNAL SQLSTATE 'ERROR'

SET MESSAGE\_TEXT = 'Ungültiger Zugriffsort!';

END IF;

END;

ELSE

SIGNAL SQLSTATE 'ERROR' SET MESSAGE\_TEXT = 'Ungültige Eingabe: Zugriffsort';

END IF;

SET @benutzer = CONCAT('\'', benutzer, '\'@\'', hostname, '\'');

SET @erstellen = CONCAT('CREATE USER ', @benutzer);

IF passwort IS NOT NULL AND passwort <> ''

THEN

SET @erstellen = CONCAT(@erstellen, ' IDENTIFIED BY \'', passwort, '\'');

END IF;

PREPARE erstellen FROM @erstellen;

EXECUTE erstellen;

DEALLOCATE PREPARE erstellen;

SET @berechtigung\_erteilen = CONCAT('GRANT ' + @berechtigungen);

SET @berechtigung\_erteilen = CONCAT(@berechtigung\_erteilen, ' ON ', @zugriffsort);

SET @berechtigung\_erteilen = CONCAT(@berechtigung\_erteilen, ' TO ', @benutzer);

PREPARE berechtigung\_erteilen FROM @berechtigung\_erteilen;

EXECUTE berechtigung\_erteilen;

DEALLOCATE PREPARE berechtigung\_erteilen;

INSERT INTO `schoolinfo\_neu`.`log\_berechtigung` (

benutzer,

timestamp,

wofuer,

typ,

berechtigung,

fuer

) VALUES (

CURRENT\_USER(),

CURRENT\_DATE(),

zugriffsort,

berechtigung,

benutzer

);

END;

# Lernende archivieren

## Kurzbeschreibung

Lernende, welche die Ausbildung beendet haben, sollen aus der Datenbank einfacher als bisher entfernt werden. Dafür werden sie mit dieser Stored Procedure in eine andere de-normalisierte Datenbank verschoben.

## Ein-/ Ausgabeparameter

|  |  |  |
| --- | --- | --- |
| Parameter | Datentyp | Beschreibung |
| klasse | VARCHAR(255) | Name der Klasse |

## Aufrufbeispiele

Mit dem Code der hier ist, kann eine Klasse ausgewählt werden, wessen Lernende anschliessend archiviert werden.

CALL sp\_archives("IAP13v");

## Code

**sp\_archives.sql**

DROP PROCEDURE IF EXISTS `sp\_archives`;

CREATE PROCEDURE `sp\_archives`(

IN klasse VARCHAR(255)

) BEGIN

SET @id\_klasse = (

SELECT `id\_klasse`

FROM `schoolinfo\_neu`.`klassen`

WHERE `name` = klasse

LIMIT 1

);

INSERT INTO `schoolinfo\_archiv`.`lernende\_archiv` (

`anrede`,

`name`,

`vorname`,

`geschlecht`,

`bm`,

`strasse`,

`ort`,

`plz`,

`richtung`,

`klasse`,

`lehrbetrieb`,

`lbstrasse`,

`lbhausnr`,

`lbplz`,

`lbort`,

`lbland`,

`note\_erf`,

`note\_knw`

) SELECT

`lernender`.`anrede`,

`lernender`.`name`,

`lernender`.`vorname`,

`lernender`.`geschlecht`,

`lernender`.`bm`,

`lernender`.`strasse`,

`lernender\_ort`.`name`,

`lernender\_ort`.`plz`,

`klasse`.`name`,

`klasse`.`beschreibung`,

`richtung`.`name`,

`lehrbetrieb`.`name`,

`lehrbetrieb`.`strasse`,

`lehrbetrieb`.`nummer`,

`lehrbetrieb\_ort`.`plz`,

`lehrbetrieb\_ort`.`name`,

`note`.`note\_erfahrung`,

`note`.`note\_knw`

FROM `schoolinfo\_neu`.`lernende` AS `lernender`

LEFT JOIN `schoolinfo\_neu`.`klassen` AS `klasse`

ON `klasse`.`id\_klasse` = `lernender`.`fk\_klasse`

LEFT JOIN `schoolinfo\_neu`.`ortschaften` AS `lernender\_ort`

ON `lernender\_ort`.`id\_ort` = `lernender`.`fk\_ort`

LEFT JOIN `schoolinfo\_neu`.`module` AS `richtung`

ON `richtung`.`id\_modul` = `lernender`.`fk\_fachrichtung`

LEFT JOIN `schoolinfo\_neu`.`lehrbetriebe` AS `lehrbetrieb`

ON `lehrbetrieb`.`id\_lehrbetrieb` = `lernender`.`fk\_lehrbetrieb`

LEFT JOIN `schoolinfo\_neu`.`ortschaften` AS `lehrbetrieb\_ort`

ON `lehrbetrieb\_ort`.`id\_ort` = `lehrbetrieb`.`fk\_ort`

LEFT JOIN `schoolinfo\_neu`.`noten` AS `note`

ON `note`.`fk\_lernender` = `lernender`.`id\_lernender`

WHERE `id\_klasse` = @id\_klasse;

DELETE FROM `schoolinfo\_neu`.`noten`

WHERE `fk\_lernender` IN (

SELECT \*

FROM (

SELECT `lernende`.`id\_lernender` AS `idl`

FROM `schoolinfo\_neu`.`noten` AS `note`

LEFT JOIN `schoolinfo\_neu`.`lernende` AS `lernender`

ON `lernender`.`id\_lernender` = `note`.`fk\_lernender`

WHERE `lernender`.`fk\_klasse` = @id\_klasse

) AS `auswahl`

);

END;

**schoolinfo\_archiv.sql**

CREATE DATABASE IF NOT EXISTS schoolinfo\_archiv;

USE schoolinfo\_archiv;

CREATE TABLE IF NOT EXISTS lernende\_archiv (

id INT PRIMARY KEY AUTO\_INCREMENT,

anrede VARCHAR(25),

name VARCHAR(50) NOT NULL,

vorname VARCHAR(50) NOT NULL,

geschlecht ENUM('M','F'),

bm TINYINT(1),

strasse VARCHAR(50),

ort VARCHAR(50),

plz VARCHAR(10),

richtung VARCHAR(50) NOT NULL,

klasse VARCHAR(10) NOT NULL,

lehrbetrieb VARCHAR(100) NOT NULL,

lbstrasse VARCHAR(50),

lbhausnr VARCHAR(10),

lbplz VARCHAR(8),

lbort VARCHAR(50),

lbland VARCHAR(15),

modulname VARCHAR(50) NOT NULL,

note\_erf DECIMAL(3, 2),

note\_knw DECIMAL(3, 2),

timestamp DATETIME NOT NULL DEFAULT current\_timestamp

);