Snippets Datenmanagement

Version: 1.1.0

Study: 2. Semester, Bachelor in Business and Computer Science

School: Hochschule Luzern - Wirtschaft

Author: Janik von Rotz (<http://janikvonrotz.ch>)

Source: <https://gist.github.com/janikvonrotz/6e27788f662fcdbba3fb>

License:

This work is licensed under the Creative Commons Attribution-ShareAlike 4.0 International License. To view a copy of this license, visit <http://creativecommons.org/licenses/by-sa/4.0/> or send a letter to Creative Commons, PO Box 1866, Mountain View, CA 94042, USA.

Inhalt

[**SQL languages** 2](#_Toc423598676)

[**Datatypes** 2](#_Toc423598677)

[**Database** 5](#_Toc423598678)

[**Table** 5](#_Toc423598679)

[**Alter Table** 5](#_Toc423598680)

[**Insert** 6](#_Toc423598681)

[**Update** 6](#_Toc423598682)

[**Delete** 6](#_Toc423598683)

[**Index** 7](#_Toc423598684)

[**Type** 7](#_Toc423598685)

[**Login** 7](#_Toc423598686)

[**User** 7](#_Toc423598687)

[**Grant/ Revoke** 8](#_Toc423598688)

[**Role** 8](#_Toc423598689)

[**Grant Role** 8](#_Toc423598690)

[**View** 9](#_Toc423598691)

[**Select** 9](#_Toc423598692)

[**Join the hard way** 10](#_Toc423598693)

[**Join the right way** 10](#_Toc423598694)

[**About Joins** 11](#_Toc423598695)

[**Select with Subqueries** 12](#_Toc423598696)

[**Union** 12](#_Toc423598697)

[**Transactions** 12](#_Toc423598698)

[**Function** 13](#_Toc423598699)

[**Procedure** 13](#_Toc423598700)

[**Variables** 14](#_Toc423598701)

[**Trigger** 14](#_Toc423598702)

# **SQL languages**

**DDL** is short name of Data Definition Language, which deals with database schemas and descriptions, of how the data should reside in the database.

**DML** is short name of Data Manipulation Language which deals with data manipulation, and includes most common SQL statements such SELECT, INSERT, UPDATE, DELETE etc, and it is used to store, modify, retrieve, delete and update data in database.

**DCL** is short name of Data Control Language which includes commands such as GRANT, and mostly concerned with rights, permissions and other controls of the database system.

**Datatypes**

Text types

| **Data type** | **Description** |
| --- | --- |
| CHAR(size) | Holds a fixed length string (can contain letters, numbers, and special characters). The fixed size is specified in parenthesis. Can store up to 255 characters |
| VARCHAR(size) | Holds a variable length string (can contain letters, numbers, and special characters). The maximum size is specified in parenthesis. Can store up to 255 characters. Note: If you put a greater value than 255 it will be converted to a TEXT type |
| TINYTEXT | Holds a string with a maximum length of 255 characters |
| TEXT | Holds a string with a maximum length of 65,535 characters |
| BLOB | For BLOBs (Binary Large OBjects). Holds up to 65,535 bytes of data |
| MEDIUMTEXT | Holds a string with a maximum length of 16,777,215 characters |
| MEDIUMBLOB | For BLOBs (Binary Large OBjects). Holds up to 16,777,215 bytes of data |
| LONGTEXT | Holds a string with a maximum length of 4,294,967,295 characters |
| LONGBLOB | For BLOBs (Binary Large OBjects). Holds up to 4,294,967,295 bytes of data |
| ENUM(x,y,z,etc.) | Let you enter a list of possible values. You can list up to 65535 values in an ENUM list. If a value is inserted that is not in the list, a blank value will be inserted.Note: The values are sorted in the order you enter them.You enter the possible values in this format: ENUM('X','Y','Z') |
| SET | Similar to ENUM except that SET may contain up to 64 list items and can store more than one choice |

Number types

| **Data type** | **Description** |
| --- | --- |
| TINYINT(size) | -128 to 127 normal. 0 to 255 UNSIGNED\*. The maximum number of digits may be specified in parenthesis |
| SMALLINT(size) | -32768 to 32767 normal. 0 to 65535 UNSIGNED\*. The maximum number of digits may be specified in parenthesis |
| MEDIUMINT(size) | -8388608 to 8388607 normal. 0 to 16777215 UNSIGNED\*. The maximum number of digits may be specified in parenthesis |
| INT(size) | -2147483648 to 2147483647 normal. 0 to 4294967295 UNSIGNED\*. The maximum number of digits may be specified in parenthesis |
| BIGINT(size) | -9223372036854775808 to 9223372036854775807 normal. 0 to 18446744073709551615 UNSIGNED\*. The maximum number of digits may be specified in parenthesis |
| FLOAT(size,d) | A small number with a floating decimal point. The maximum number of digits may be specified in the size parameter. The maximum number of digits to the right of the decimal point is specified in the d parameter |
| DOUBLE(size,d) | A large number with a floating decimal point. The maximum number of digits may be specified in the size parameter. The maximum number of digits to the right of the decimal point is specified in the d parameter |
| DECIMAL(size,d) | A DOUBLE stored as a string , allowing for a fixed decimal point. The maximum number of digits may be specified in the size parameter. The maximum number of digits to the right of the decimal point is specified in the d parameter |

Date types

| **Data type** | **Description** |
| --- | --- |
| DATE() | A date. Format: YYYY-MM-DDNote: The supported range is from '1000-01-01' to '9999-12-31' |
| DATETIME() | \*A date and time combination. Format: YYYY-MM-DD HH:MI:SSNote: The supported range is from '1000-01-01 00:00:00' to '9999-12-31 23:59:59' |
| TIMESTAMP() | \*A timestamp. TIMESTAMP values are stored as the number of seconds since the Unix epoch ('1970-01-01 00:00:00' UTC). Format: YYYY-MM-DD HH:MI:SSNote: The supported range is from '1970-01-01 00:00:01' UTC to '2038-01-09 03:14:07' UTC |
| TIME() | A time. Format: HH:MI:SSNote: The supported range is from '-838:59:59' to '838:59:59' |
| YEAR() | A year in two-digit or four-digit format.Note: Values allowed in four-digit format: 1901 to 2155. Values allowed in two-digit format: 70 to 69, representing years from 1970 to 2069 |

**Database**

Create

create database dbname;

Drop

drop database dbname;

**Table**

Check if not exit and create

IF OBJECT\_ID('tbl\_kunde', N'U') is not null

drop table tbl\_kunde;

GO

create table tbl\_kunde (

id\_kunde int not null primary key,

fi\_moral\_nr int,

name varchar(25) not null,

vorname varchar not null,

wohnort varchar

);

GO

**Alter Table**

Primary Key

ALTER TABLE tbl\_kunde ADD PRIMARY KEY (id\_kunde);

Foreign Key

ALTER TABLE tbl\_kunde ADD CONSTRAINT FK\_fi\_moral\_nr FOREIGN KEY (fi\_moral\_nr)

REFERENCES tkey\_moral

ON UPDATE CASCADE

ON DELETE SET NULL;

Constraint

ALTER TABLE tkey\_moral ADD CONSTRAINT PK\_id\_moral\_nr PRIMARY KEY (id\_moral\_nr);

ALTER TABLE tbl\_kunde ADD CONSTRAINT FK\_fi\_moral\_nr FOREIGN KEY (fi\_moral\_nr)

REFERENCES tkey\_moral

ON UPDATE CASCADE

ON DELETE SET NULL;

**Insert**

Selected fields

insert into tkey\_moral (id\_moral\_nr, moral\_bez) values (1, 'gut'), (2, 'schlecht'), (3, 'schlecht');

All fields

INSERT INTO tbl\_kunde VALUES (3838,1,'Meier','Laura','Waldibrücke')

**Update**

Update by condition

update tbl\_kunde set name = 'Menzer' where name = 'Waltenspühl-Menzer'

update tass\_police set praem\_stufe = 101 where praem\_stufe = 108

**Delete**

All

delete from tbl\_kunde

Condition

delete from tkey\_moral where id\_moral\_nr = 4

delete from tbl\_kunde where vorname = 'Peter' and name = 'Fischer' or vorname = 'Martin' and name = 'Müller'

**Index**

Create

create unique index ix\_kund\_name on tbl\_kunde (name)

Disable

alter index ix\_kund\_name on tbl\_kunde disable

Rebuild

alter index ix\_kund\_name on tbl\_kunde rebuild

Reorganize

alter index ix\_kund\_name on tbl\_kunde reorganize

Drop

drop index ix\_kund\_name on tbl\_kunde

Alter

drop index ix\_kund\_name on tbl\_kunde

**Type**

create

create type tp\_moralisches from numeric(9,0)

**Login**

change password

alter login stud23 with password = 'hello' old\_password = 'pass\_wd23'

**User**

create

create user romulus from login romulus

drop

drop user romulus

**Grant/ Revoke**

Available permissions: CREATE DEFAULT, CREATE FUNCTION, CREATE PROCEDURE, CREATE ROLE, CREATE TABLE, CREATE TYPE, CREATE VIEW, DELETE, EXECUTE, INSERT, SELECT, UPDATE

Grant rights

grant select, insert, delete, references, update to romulus

Grant refrences

grant references (id\_moral\_nr) on tkey\_moral to romulus

Revoke

revoke insert, delete, references, update to romulus

**Role**

Create

create role verkauf

Add member

exec sp\_addrolemember 'verkauf', 'anna'

**Grant Role**

Grant rights

grant select, update, insert, delete on tbl\_kunde to verkauf

**View**

Create detailed

CREATE VIEW v\_kunden (id, name, vorname, ort, police, gebiet) AS

SELECT k.id\_kunde, k.name, k.vorname, k.wohnort, v.vers\_bez,

v.vers\_gebiet

FROM tbl\_kunde AS k, tkey\_versicherung AS v, tass\_police AS p

WHERE k.id\_kunde = p.id\_fi\_kunde

AND p.id\_fi\_vers\_art = v.id\_vers\_art;

Create simple

create view plain as select id\_kunde, name, vorname from tbl\_kunde

Create advanced

create view v\_umsatz as select sum(cast(bezahlt as float)) as 'Umsatz aus Policen' from tass\_police

**Select**

Basic

select dt\_jahr from dbo.tbl\_stueck

Condition

select dt\_jahr from dbo.tbl\_stueck where dt\_jahr is not null

Order

select dt\_jahr from dbo.tbl\_stueck

where dt\_jahr is not null

order by dt\_jahr desc

Distinct

select distinct dt\_jahr from dbo.tbl\_stueck

where dt\_jahr is not null

order by dt\_jahr asc

And Condition

select \* from tbl\_stueck where dt\_jahr = 1970 and dt\_zeit > 3

Between

select \* from tbl\_stueck where dt\_jahr = 1970 and dt\_zeit between 3 and 10

Like

select \* from tbl\_stueck where dt\_stueck\_titel like 'Let%'

Count

select count(dt\_stueck\_titel) from tbl\_stueck where dt\_stueck\_titel like 'Let%'

Having and without join

select i.dt\_name, count(s.dt\_stueck\_titel) from tbl\_stueck as s, tkey\_interpret as i

where s.fi\_interpret = i.id\_interpret

group by i.dt\_name

having count(s.dt\_stueck\_titel) >10

order by count(s.dt\_stueck\_titel) de

**Join the hard way**

Inner Join

select s.dt\_stueck\_titel, i.dt\_name

from tbl\_stueck as s, tkey\_interpret as i

where s.fi\_interpret = i.id\_interpret

order by s.dt\_stueck\_titel

Multi Join

select sa.id\_jahr, st.dt\_stueck\_titel, ip.dt\_name

from tbl\_stueck as st, tass\_stueck\_sampler as ss, tkey\_sampler as sa, tkey\_interpret as ip

where ss.id\_fi\_stueck\_nr = st.id\_stueck\_nr

and ss.id\_fi\_jahr = sa.id\_jahr

and st.fi\_interpret = ip.id\_interpret

order by st.dt\_stueck\_titel

**Join the right way**

Inner Equi Key Joining

select dt\_stueck\_titel, dt\_name

from tbl\_stueck join tkey\_interpret

on fi\_interpret = id\_interpret

order by dt\_stueck\_titel

Multi Inner Equi Key Joining

select id\_fi\_jahr, dt\_stueck\_titel, dt\_name

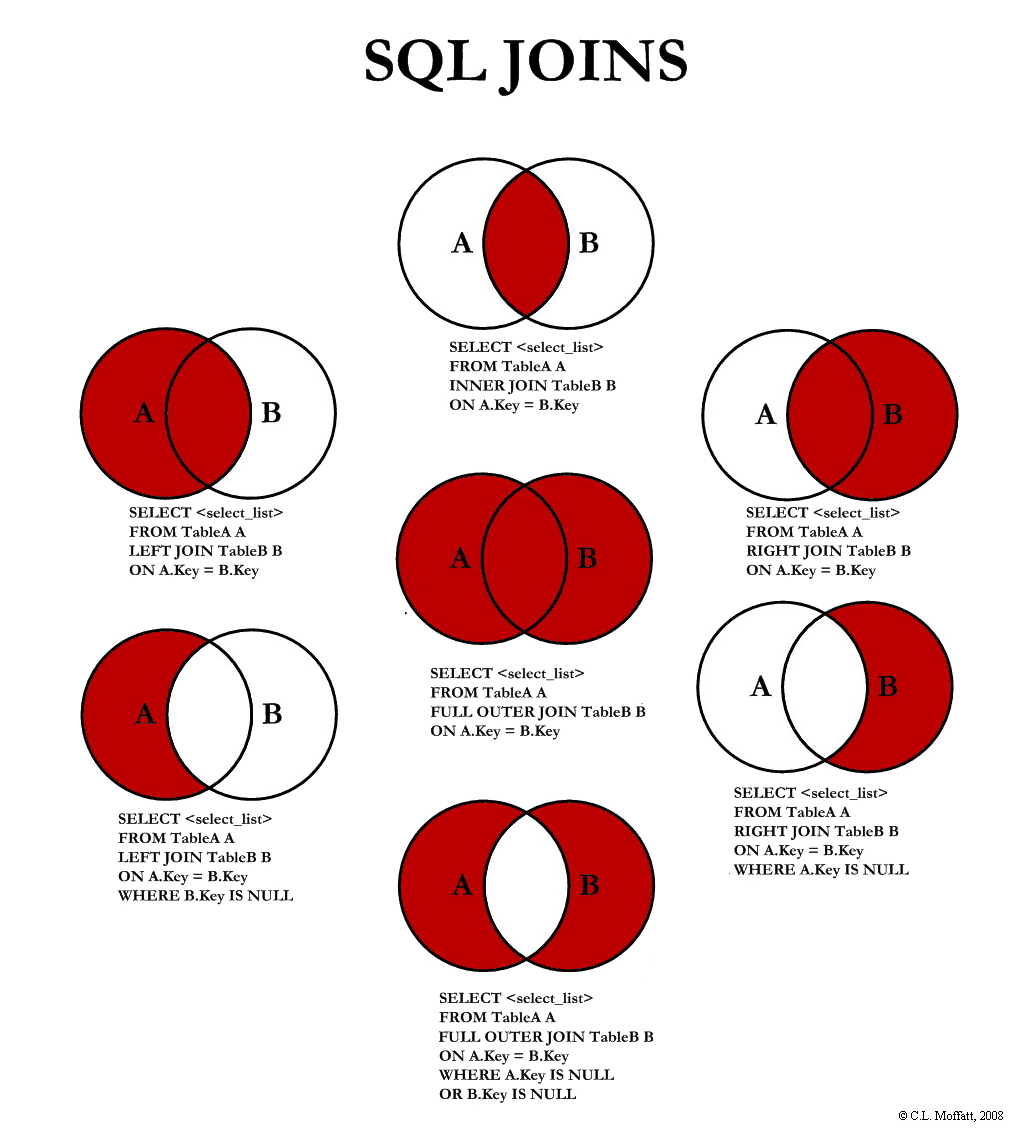
from tbl\_stueck

join tass\_stueck\_sampler on id\_fi\_stueck\_nr = id\_stueck\_nr

join tkey\_interpret on fi\_interpret = id\_interpret

order by dt\_stueck\_titel

**About Joins**

[](https://camo.githubusercontent.com/a2ceb7ca016949dd2b9094d04761ef87e847a9e4/687474703a2f2f692e737461636b2e696d6775722e636f6d2f687a6c38652e706e67)

SELECT customers.id, customers.name, items.name, customers.state

FROM customers, items

WHERE customers.id=seller\_id

ORDER BY customers.id

**Join tables** -> Joining two tables together in a query output. The third line is important because it shows how the two tables are related (in this case it is their key values).

SELECT customers.name, items.name FROM customers

LEFT OUTER JOIN items ON customers.id=seller\_id

**LEFT/RIGHT OUTER JOIN** -> Takes the table left of the word 'LEFT' or 'RIGHT' (in this case customers) and joins it regardless of whether it has any values or not. So the above statement shows all users/customers, even if they aren't selling anything.

**Select with Subqueries**

Select max and min values

select dt\_stueck\_titel as Titel, dt\_zeit as Zeit

from tbl\_stueck

where dt\_zeit = (select max(dt\_zeit) from tbl\_stueck)

or dt\_zeit = (select min(dt\_zeit) from tbl\_stueck)

order by dt\_zeit;

Select with query in condition

select dt\_stueck\_titel as Titel, dt\_zeit as Zeit

from tbl\_stueck

where dt\_zeit between (select avg(dt\_zeit) from tbl\_stueck)\*0.9

and (select avg(dt\_zeit) from tbl\_stueck)\*1.1

order by dt\_zeit;

Select query as value

select dt\_stueck\_titel as Titel,

dt\_zeit/(select avg(dt\_zeit) from tbl\_stueck)\*100 as Zeit

from tbl\_stueck

where dt\_stueck\_titel = 'You Shook Me'

**Union**

Unify two result sets with a condition

select \* from

(select dt\_stueck\_titel as titel, 'stück' as 'type' from tbl\_stueck

union

select dt\_name as titel, 'inter' as 'type' from tkey\_interpret

union

select dt\_stao as titel, 'stao' as 'type' from tkey\_standort) as t

where t.titel like '%boy%'

**Transactions**

Run a transaction

begin transaction

update tass\_police set bezahlt = 0 where id\_fi\_kunde = 3533 and id\_fi\_vers\_art = 1700

commit

-- or rollback

**Function**

A function can be called from inside a statement just like any other function and can return a scalar value.

Create - Get value from table

create function f\_plic\_bez()

returns decimal(10,2)

as begin

return (select sum(bezahlt) from tass\_police)

end;

go

select dbo.f\_plic\_bez() AS 'Summe aller bezahlten Leistungen'

go

Drop

drop function f\_bezahlt\_versich

Create - With parameters

create function f\_rabatt(@name varchar(40), @vers varchar(30))

returns int

as begin

return (select (praem\_stufe-100)\*10 from tass\_police

join tbl\_kunde on id\_fi\_kunde = id\_kunde

join tkey\_versicherung on id\_fi\_vers\_art = id\_vers\_art

where name = @name and vers\_bez = @vers)

end

**Procedure**

Stored procedures are stored as precompilated code (stored routine) and called by the programmer wherever it wants to fire. Stored procedure can return value(s).

Create and execute

CREATE PROCEDURE p\_polic\_del @fname VARCHAR(30), @versich VARCHAR(30)

AS

BEGIN

DELETE FROM tass\_police

FROM tbl\_kunde, tkey\_versicherung

WHERE name = @fname

AND id\_kunde = id\_fi\_kunde

AND id\_fi\_vers\_art = id\_vers\_art

AND vers\_bez = @versich

IF @@ROWCOUNT = 0

PRINT 'Police existiert nicht.'

ELSE PRINT 'Löschung vollzogen.'

END

GO

EXECUTE p\_polic\_del 'Meier', 'Taggeld';

**Variables**

Declare

DECLARE @veraenderung SMALLINT = 180;

DECLARE @neue\_summe INT;

**Trigger**

Triggers are named database objects fired automatically when insert, delete, update (or other event) occurred, there can be no explicit invocation. Trigger can not return any data.

Create simple

create trigger t\_ort on tbl\_kunde

after insert, update

as beginn

set @ort = select wohnort from inserted)

if((@ort like '/^[A-Z]') && len(@ort) < 2)

rollback transaction

end

end

Drop

drop trigger t\_ort

Checks the referential integrity

create trigger t\_bst\_mut on bst

for insert, update

as

begin

set nocount on;

if(select id\_fi\_k from inserted) NOT IN (select id\_kund from knd)

begin

raiserror('Es besteht kein entsprechender Kunde.', 15, 1);

rollback transaction;

end

end;

Replaces on delete no action

create trigger t\_ku\_del on knd

for delete

as

begin

set nocount on;

if((select id\_kunde from deleted) IN (select id\_fi\_k from bst))

begin

raiserror('Löschung verwert; es bestehen noch Bestelungen für diesen Kunden.', 15, 1)

rollback transaction

end

end;

Replaces on update cascade

create trigger t\_update\_knd on knd

after update

as

begin

set nocount on;

update bst set id\_fi\_k = (select id\_kund from inserted)

end;

Replaces on delete cascade

create trigger t\_delete\_knd on knd

after delete

as

begin

delete from bst where id\_fi\_k = (select id\_kund from deleted)

end;