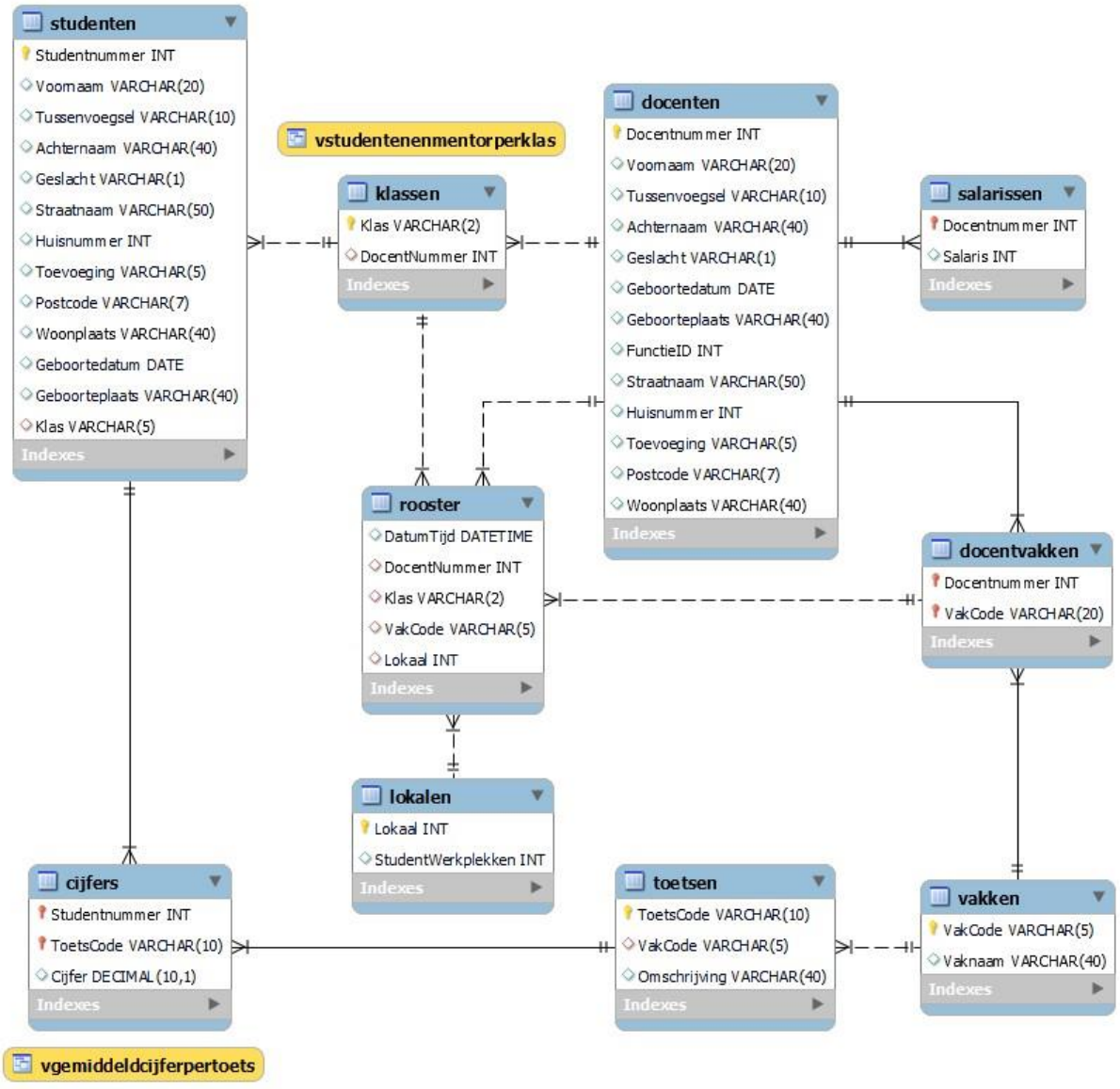
MBO_ICT database

In de database MBO-ICT staan de gegevens van een MBO school voor ICT. In de tabellen staan de gegevens van alle studenten, docenten, klassen, lokalen, vakken, toetsen, cijfers, docentvakken, docentsalarissen en een lessenrooster volgens onderstaand Entity Relation Diagram.



De database MBO-ICT bevat de tabellen zoals weergegeven in onderstaande tabel.

Tabel	Omschrijving, kolommen en constraints
Lokalen	Deze tabel bevat de gegevens van alle lokalen binnen de school. Kolommen: • **Lokaal** (elk lokaal heeft een uniek lokaalnummer) • **StudentWerkplekken** (het aantal studentwerkplekken in het lokaal)
Docenten	Deze tabel bevat de gegevens van de docenten. Kolommen: • **Docentnummer** (elke docent heeft een uniek docentnummer) • **Voornaam** (voornaam van de docent) • **Tussenvoegsel** (tussenvoegsel tussen voornaam en achternaam) • **Achternaam** (achternaam van de docent) • **Geslacht** (geslacht van de docent) • **Geboortedatum** (geboortedatum van de docent) • **Geboorteplaats** (geboorteplaats van de docent) • **FunctieID** (functiecode van de docent) • **Straatnaam** (straatnaam van de docent)) • **Huisnummer** (huisnummer van de docent) • **Toevoeging** (toevoeging bij huisnummer) • **Postcode** (postcode van de docent) • **Woonplaats** (woonplaats van de docent)
Klassen	Deze tabel bevat de gegevens van de klassen. Kolommen: • **Klas** (klascode) • **DocentNummer** (docentnummer van klassenmentor)
Studenten	Deze tabel bevat de gegevens van alle studenten binnen de school. Kolommen: • **Studentnummer** (elke student heeft een uniek studentnummer) • **Voornaam** (voornaam van de student) • **Tussenvoegsel** (tussenvoegsel tussen voornaam en achternaam) • **Achternaam** (achternaam van de student) • **Geslacht** (geslacht van de student) • **Straatnaam** (straatnaam van de student) • **Huisnummer** (huisnummer van de student) • **Toevoeging** (toevoeging bij huisnummer) • **Postcode** (postcode van de student) • **Woonplaats** (woonplaats van student) • **Geboortedatum** (geboortedatum van student) • **Geboorteplaats** (geboorteplaats van student) • **Klas** (klas van student)
Vakken	Deze tabel bevat de gegevens over de lesvakken binnen de school. Kolommen: • **VakCode** (code van het lesvak) • **Vaknaam** (volledige naam van het lesvak)

DocentVakken	Deze tabel bevat gegevens over de lesvakken die de docenten kunnen geven. Kolommen: • **Docentnummer** (unieke docentnummer) • **VakCode** (lesvak dat docent kan geven)
Salarissen	Deze tabel bevat de salarissen van de docenten. Deze tabel mag zeker niet door alle medewerkers worden ingezien of worden aangepast! Kolommen: • **Docentnummer** (unieke docentnummer) • **Salaris** (maandelijkse salaris van de docent)
Rooster	Deze tabel bevat de gegevens van het lesrooster van de docenten en studenten. Kolommen: • **DatumTijd** (datum en tijd van lesuur) • **DocentNummer** (unieke docentcode) • **Klas** (unieke klascode) • **VakCode** (unieke Vakcode) • **Lokaal** (unieke Lokaalcode)
Toetsen	Deze tabel bevat de gegevens van de toetsen binnen de school. Kolommen: • **ToetsCode** (unieke code van de toets) • **VakCode** (unieke code van het lesvak) • **Omschrijving** (omschrijving van de toets)
Cijfers	Deze tabel bevat de gegevens van de behaalde cijfers voor toetsen door studenten. Kolommen: • **Studentnummer** (unieke nummer van de student) • **ToetsCode** (code van de toets) • **Cijfer** (behaalde cijfer)

In de database MBO_ICT zijn de volgende constraints gedefinieerd:

Lokalen	• *De kolom **Lokaal** heeft een **Primary Key constraint** waardoor deze kolom niet leeg kan zijn en ook géén dubbele waarde kan bevatten.*
Docenten	• *De kolom **Docentnummer** heeft een **Primary Key constraint** waardoor deze kolom niet leeg kan zijn en ook géén dubbele waarde kan bevatten.*
Klassen	• *De kolom **Klas** heeft een **Primary Key constraint** waardoor deze kolom niet leeg kan zijn en ook géén dubbele waarde kan bevatten.* • *De kolom **DocentNummer** heeft een **Foreign Key constraint** naar kolom DocentNummer in tabel Docenten waardoor deze kolom alleen waarden kan bevatten die reeds voorkomen in de kolom Docent van tabel Docenten.*

Studenten	• *De kolom **Studentnummer** heeft een **Primary Key constraint** waardoor deze kolom niet leeg kan zijn en ook géén dubbele waarde kan bevatten.* • *De kolom **Klas** heeft een **Foreign Key constraint** naar kolom Klas in tabel Klassen waardoor deze kolom alleen waarden kan bevatten die reeds voorkomen in de kolom Klas van tabel Klassen.*
Vakken	• *De kolom **VakCode** heeft een **Primary Key constraint** waardoor deze kolom niet leeg kan zijn en ook géén dubbele waarde kan bevatten.*
DocentVakken	• *De combinatie van **Docentnummer** en **Vakcode** heeft een **Primary Key constraint** waardoor deze kolommen niet leeg kunnen zijn en de combinatie van Docentnummer en Vakcode ook géén dubbele waarde kan bevatten. Let wel: Docentnummer en Vakcode kunnen afzonderlijk wél meerdere malen voorkomen, echter de combinatie ervan niet!* • *De kolom **VakCode** heeft een **Foreign Key constraint** naar kolom VakCode in tabel Vakken waardoor deze kolom alleen waarden kan bevatten die reeds voorkomen in de kolom VakCode van tabel Vakken.* • *De kolom **Docentnummer** heeft een **Foreign Key constraint** naar kolom Docentnummer in tabel Docenten waardoor deze kolom alleen waarden kan bevatten die reeds voorkomen in de kolom Docentnummer van tabel Docenten.*
Salarissen	• *De kolom **Docentnummer** heeft een **Primary Key constraint** waardoor deze kolom niet leeg kan zijn en ook géén dubbele waarde kan bevatten.* • *De kolom **Docentnummer** heeft een **Foreign Key constraint** naar kolom Docentnummer in tabel Docenten waardoor deze kolom alleen waarden kan bevatten die reeds voorkomen in de kolom Docentnummer van tabel Docenten.*
Rooster	• *De combinatie van **DatumTijd** en **DocentNummer** heeft een **Primary Key constraint** waardoor deze kolommen niet leeg kunnen zijn en de combinatie van DatumTijd en DocentNummer ook géén dubbele waarde kan bevatten. Let wel: DatumTijd en DocentNummer kunnen afzonderlijk wél meerdere malen voorkomen, echter de combinatie ervan niet!* • *De combinatie van **DatumTijd** en **Klas** heeft een **Primary Key constraint** waardoor deze kolommen niet leeg kunnen zijn en de combinatie van DatumTijd en Klas ook géén dubbele waarde kan bevatten. Let wel: DatumTijd en Klas kunnen afzonderlijk wél meerdere malen voorkomen, echter de combinatie ervan niet!* • *De combinatie van **DatumTijd** en **Lokaal** heeft een **Primary Key constraint** waardoor deze kolommen niet leeg kunnen zijn en de combinatie van DatumTijd en Lokaal ook géén dubbele waarde kan*

	bevatten. Let wel: DatumTijd en Lokaal kunnen afzonderlijk wél meerdere malen voorkomen, echter de combinatie ervan niet! • *De kolom **Lokaal** heeft een **Foreign Key constraint** naar kolom Lokaal in tabel Lokalen waardoor deze kolom alleen waarden kan bevatten die reeds voorkomen in de kolom Lokaal van tabel Lokalen.* • *De kolom **DocentNummer** heeft een **Foreign Key constraint** naar kolom DocentNummer in tabel Docenten waardoor deze kolom alleen waarden kan bevatten die reeds voorkomen in de kolom DocentNummer van tabel Docenten.* • *De kolom **Klas** heeft een **Foreign Key constraint** naar kolom Klas in tabel Klassen waardoor deze kolom alleen waarden kan bevatten die reeds voorkomen in de kolom Klas van tabel Klassen.* • *De combinatie van **Docentnummer** en **Vakcode** heeft een **Primary Key constraint** waardoor deze kolommen niet leeg kunnen zijn en de combinatie van Docentnummer en Vakcode ook géén dubbele waarde kan bevatten. Let wel: Docentnummer en Vakcode kunnen afzonderlijk wél meerdere malen voorkomen, echter de combinatie ervan niet!*
Toetsen	• *De kolom **ToetsCode** heeft een **Primary Key constraint** waardoor deze kolom niet leeg kan zijn en ook géén dubbele waarde kan bevatten.* • *De kolom **VakCode** heeft een **Foreign Key constraint** naar kolom VakCode in tabel Vakken waardoor deze kolom alleen waarden kan bevatten die reeds voorkomen in de kolom VakCode van tabel Vakken.*
Cijfers	• *De combinatie van **Studentnummer** en **ToetsCode** heeft een **Primary Key constraint** waardoor deze kolommen niet leeg kunnen zijn en de combinatie van Studentnummer en ToetsCode ook géén dubbele waarde kan bevatten. Let wel: Studentnummer en ToetsCode kunnen afzonderlijk wél meerdere malen voorkomen, echter de combinatie ervan niet!* • *De kolom **Studentnummer** heeft een **Foreign Key constraint** naar kolom Studentnummer in tabel Studenten waardoor deze kolom alleen waarden kan bevatten die reeds voorkomen in de kolom Studentnummer van tabel Studenten.* • *De kolom **ToetsCode** heeft een **Foreign Key constraint** naar kolom ToetsCode in tabel Toetsen waardoor deze kolom alleen waarden kan bevatten die reeds voorkomen in de kolom ToetsCode van tabel Toetsen.*

Basisopdrachten in SQL

Opdracht	Syntaxis	Beschrijving
ALTER TABLE ADD column	**ALTER TABLE table_name ADD column_name datatype;**	Het wordt gebruikt om kolommen toe te voegen aan een tabel in een database
ALTER TABLE ADD constraint	**ALTER TABLE table_name ADD CONSTRAINT constraint_name PRIMARY KEY (c1,c2);** **ALTER TABLE table_name ADD CONSTRAINT constraint_name FOREIGN KEY (c1,c2) REFERENCES table_name (c1,c2);**	Het wordt gebruikt om een Primaire sleutel op een tabel te plaatsen. Het wordt gebruikt om een Verwijzende sleutel op een tabel te plaatsen.
AND	**SELECT column_name(s) FROM table_name WHERE column_1 = value_1 AND column_2 = value_2;**	Het is een operator die wordt gebruikt om twee voorwaarden te combineren
AS	**SELECT column_name AS 'Alias' FROM table_name;**	Het is een trefwoord in SQL dat wordt gebruikt om een kolom of tabel te hernoemen met een aliasnaam
AVG	**SELECT AVG(column_name) FROM table_name;**	Het wordt gebruikt om een numerieke kolom samen te voegen en het gemiddelde te retourneren
BETWEEN	**SELECT column_name(s) FROM table_name WHERE column_name BETWEEN value_1 AND value_2;**	Het is een bewerking die wordt gebruikt om het resultaat binnen een bepaald bereik te filteren
CASE	**SELECT column_name, CASE WHEN condition THEN 'Result_1' WHEN condition THEN 'Result_2' ELSE 'Result_3' END FROM table_name;**	Het is een instructie die wordt gebruikt om verschillende uitvoer binnen een SELECT-instructie te creëren
COUNT	**SELECT COUNT(column_name) FROM table_name;**	Het is een functie die de naam van een kolom als argument neemt en het aantal rijen telt wanneer de kolom niet NULL is
CREATE DATABASE	**CREATE DATABASE database_name;**	Het wordt gebruikt om een nieuwe database aan te maken
CREATE TABLE	**CREATE TABLE table_name (column_1 datatype, column_2 datatype, column_3 datatype);**	Het wordt gebruikt om een nieuwe tabel in een database te maken en de naam van de tabel en de kolommen erin op te geven
CREATE VIEW	**CREATE VIEW view_name AS SELECT column_name(2) FROM t1 WHERE column_name operator value;**	Het wordt gebruikt om een view aan te maken op kolommen van een of meerdere tabellen.

DELETE	**DELETE FROM table_name WHERE some_column = some_value;**	Het wordt gebruikt om de rijen uit een tabel te verwijderen
DROP DATABASE	**DROP DATABASE database_name;**	Het wordt gebruikt om een database te verwijderen.
DROP TABLE	**DROP TABLE table_name;**	Het wordt gebruikt om een tabel te verwijderen.
GROUP BY	**SELECT column_name, COUNT(*) FROM table_name GROUP BY column_name;**	Het is een clausule in SQL die wordt gebruikt voor statistische functies in samenwerking met de SELECT-instructie
HAVING	**SELECT column_name, COUNT(*) FROM table_name GROUP BY column_name HAVING COUNT(*) > value;**	Het wordt gebruikt in SQL omdat het sleutelwoord WHERE niet kan worden gebruikt in aggregatiefuncties
IN	**SELECT column_name(s) FROM table_name WHERE column_name IN (value_1, value_2,...);**	Het is een bewerking die wordt gebruikt om het resultaat binnen een bepaalde reeks te filteren
INNER JOIN	**SELECT column_name(s) FROM table_1 JOIN table_2 ON table_1.column_name = table_2.column_name;**	Het wordt gebruikt om rijen uit verschillende tabellen te combineren als de Join-voorwaarde TRUE wordt
INSERT	**INSERT INTO table_name (column_1, column_2, column_3) VALUES (value_1, 'value_2', value_3);**	Het wordt gebruikt om nieuwe rijen aan een tabel toe te voegen
IS NULL/ IS NOT NULL	**SELECT column_name(s) FROM table_name WHERE column_name IS NULL;**	Het is een operator die wordt gebruikt met de WHERE-component om te controleren op lege waarden
LIKE	**SELECT column_name(s) FROM table_name WHERE column_name LIKE 'pattern';**	Het is een speciale operator die wordt gebruikt met de WHERE-component om naar een specifiek patroon in een kolom te zoeken
LIMIT	**SELECT column_name(s) FROM table_name LIMIT number;**	Het is een clausule om het maximale aantal rijen op te geven dat de resultatenset moet hebben
MAX	**SELECT MAX(column_name) FROM table_name;**	Het is een functie die een aantal kolommen als argument neemt en de grootste waarde daarvan retourneert
MIN	**SELECT MIN(column_name) FROM table_name;**	Het is een functie die een aantal kolommen als argument neemt en de kleinste waarde ervan retourneert
OR	**SELECT column_name**	Het is een operator die wordt gebruikt om de resultaatset te filteren zodat deze

	FROM table_name WHERE column_name = value_1 OR column_name = value_2;	alleen de rijen bevat waarin een van beide voorwaarden WAAR is
ORDER BY	**SELECT column_name FROM table_name ORDER BY column_name ASC | DESC;**	Het is een clausule die wordt gebruikt om de resultaatset op een bepaalde kolom te sorteren, hetzij numeriek of alfabetisch
OUTER JOIN	**SELECT column_name(s) FROM table_1 LEFT JOIN table_2 ON table_1.column_name = table_2.column_name;**	Het is uitgegeven om rijen uit verschillende tabellen te combineren, zelfs als de voorwaarde NIET WAAR is
ROUND	**SELECT ROUND(column_name, integer) FROM table_name;**	Het is een functie die de kolomnaam en een geheel getal als argument neemt en de waarden in een kolom afrondt op het aantal decimalen gespecificeerd door een geheel getal
SELECT	**SELECT column_name FROM table_name;**	Het is een instructie die wordt gebruikt om gegevens uit een database op te halen
SELECT DISTINCT	**SELECT DISTINCT column_name FROM table_name;**	Het wordt gebruikt om aan te geven dat de instructie een query is die unieke waarden in opgegeven kolommen retourneert
SUM	**SELECT SUM(column_name) FROM table_name;**	Het is een functie die wordt gebruikt om de som van waarden uit een bepaalde kolom te retourneren
Sub-query	**SELECT column_name(s) FROM table_name WHERE column_name IN (SELECT column_name FROM table_name);**	Het resultaat uit de sub-query wordt gebruikt in de WHERE van de omliggende query.
UPDATE	**UPDATE table_name SET some_column = some_value WHERE some_column = some_value;**	Het wordt gebruikt om rijen in een tabel te bewerken
USE	**USE database_name;**	Het wordt gebruikt om een database courant te maken.
WHERE	**SELECT column_name(s) FROM table_name WHERE column_name operator value;**	Het is een clausule die wordt gebruikt om de resultaatset te filteren om de rijen op te nemen waarvan de voorwaarde TRUE is
WITH	**WITH temporary_name AS (SELECT * FROM table_name) SELECT * FROM temporary_name WHERE column_name operator value;**	Het wordt gebruikt om het resultaat van een bepaalde query op te slaan in een tijdelijke tabel met behulp van een alias

Opdrachten en syntaxis voor het opvragen van gegevens uit een enkele tabel en meerdere tabellen:

Enkele tabel	Meerdere tabellen
SELECT column_name(s) **FROM t1** **WHERE column_name1 IN (SELECT column_name2 FROM t1);** Om met een subquery gegevens uit tabel t1 te selecteren waarbij de kolom1 gelijk is aan de waarde uit kolom2 uit dezelfde tabel t1.	**SELECT column_name(s)** **FROM t1** **WHERE column_name IN (SELECT column_name FROM t2);** Om met een subquery gegevens uit tabel t1 te selecteren waarvoor geldt dat de kolom gelijk is aan de kolomwaarde uit de selectie uit t2.
SELECT c1 **FROM t;** Om de gegevens in kolom c1 uit tabel t te selecteren	**SELECT c1, c2 FROM t1 INNER JOIN t2 on condition;** Selecteer kolom c1 en c2 uit tabel t1 en voer een inner join uit tussen t1 en t2
SELECT * **FROM t;** Om alle rijen en kolommen uit tabel t te selecteren	**SELECT c1, c2 FROM t1 LEFT JOIN t2 on condition;** Selecteer kolom c1 en c2 uit tabel t1 en voer een linkse join uit tussen t1 en t2
SELECT c1 **FROM t** **WHERE c1 = 'test';** Om gegevens in kolom c1 uit tabel t te selecteren, waarbij c1=test	**SELECT c1, c2** **FROM t1** **RIGHT JOIN t2 on condition;** Selecteer kolom c1 en c2 uit tabel t1 en voer een rechter join uit tussen t1 en t2
SELECT c1 **FROM t** **ORDER BY c1 ASC (DESC);** Gegevens in kolom c1 uit tabel t selecteren in oplopende of aflopende volgorde	**SELECT c1, c2 FROM t1 FULL OUTER JOIN t2 on condition;** Selecteer kolom c1 en c2 uit tabel t1 en voer een volledige outer join uit tussen t1 en t2 MySQL variant FULL JOIN: **SELECT c1, c2 FROM t1 LEFT JOIN t2 on condition** **UNION** **SELECT c1, c2 FROM t1 RIGHT JOIN t2 on condition;**
SELECT c1 **FROM t** **ORDER BY c1LIMIT n OFFSET** **Offset;** Om de offset van rijen over te slaan en de volgende n rijen te retourneren	**SELECT c1, c2** **FROM t1** **CROSS JOIN t2;** Selecteer kolom c1 en c2 uit tabel t1 en produceer een Cartesiaans product van rijen in een tabel
SELECT c1, aggregate(c2) **FROM t** **GROUP BY c1;** Om rijen te groeperen met behulp van een aggregatiefunctie	**SELECT c1, c2** **FROM t1, t2;** Selecteer kolom c1 en c2 uit tabel t1 en produceer een Cartesiaans product van rijen in een tabel
SELECT c1, aggregate(c2) **FROM t** **GROUP BY c1HAVING condition;** Groepeer rijen met behulp van een aggregatiefunctie en filter deze groepen met behulp van de 'HAVING'-clausule	**SELECT c1, c2** **FROM t1 A** **INNER JOIN t2 B on condition;** Selecteer kolom c1 en c2 uit tabel t1 en koppel deze aan zichzelf met behulp van de INNER JOIN-clausule

Database Beheeropdrachten in MySQL

Opdracht	Syntaxis	Beschrijving
SELECT user,host FROM mysql.user;	**SELECT user,host FROM mysql.user;**	Maakt een overzicht van alle aanwezige database gebruikers en rollen.
CREATE USER '<*username*>'@'<*FromHost*>' IDENTIFIED BY '<*password*>';	**CREATE USER 'User1'@'localhost' IDENTIFIED BY 'schoolvoorict';** **CREATE USER 'User2'@'192.168.68.110' IDENTIFIED BY 'schoolvoorict';** **CREATE USER 'User3'@'192.168.68' IDENTIFIED BY 'schoolvoorict';** **CREATE USER 'User4'@'%' IDENTIFIED BY 'schoolvoorict';**	Aanmaken van nieuwe gebruiker in MySQL. @'localhost' = gebruiker ingelogd op computer waaropde MySQL database geinstalleerd is. @'192.168.68.110' = gebruiker ingelogd op specifiek ip-adres @'192.168.68' = gebruiker ingelogd op host binnen subnet @'%' = gebruiker ingelogd vanuit willekeurige host op netwerk.
GRANT [SELECT|INSERT|UPDATE|DELETE] ON <tabelnaam> TO '<gebruikersnaam>'@'<FromHost>';	**GRANT SELECT ON Salarissen TO 'User1'@'localhost';** **GRANT SELECT, INSERT ON Docenten TO 'User2'@'192.168.68.110';** **GRANT SELECT, INSERT, UPDATE ON Toetsen TO 'User3'@'192.168.68';** **GRANT SELECT, INSERT, UPDATE, DELETE ON Studenten TO 'User4'@'%';**	Rechten toekennen op tabellen aan gebruikers.
SHOW GRANTS FOR '<user>'@'<host>';	**SHOW GRANTS FOR 'User1'@'localhost';**	Bekijken van de rechten van gebruikers op tabellen.
REVOKE [SELECT|INSERT|UPDATE|DELETE] ON <tabelnaam> FROM '<gebruikersnaam>'@'<FromHost>';	**REVOKE SELECT ON Salarissen FROM 'User1'@'localhost';** **REVOKE SELECT, INSERT ON Docenten FROM 'User2'@'192.168.68.110';** **REVOKE SELECT, INSERT, UPDATE ON Toetsen FROM 'User3'@'192.168.68';** **REVOKE SELECT, INSERT, UPDATE, DELETE ON Studenten FROM 'User4'@'%';**	Rechten op tabellen afnemen van gebruikers.

DROP USER '<*username*>'@'<*FromHost*>';	**DROP USER 'User1'@'localhost';**	Verwijderen van gebruikers
CREATE ROLE <role_name>;	**CREATE ROLE MBO_read;**	Maakt een nieuwe rol (MBO_read) aan om privileges aan toe te kennen.
GRANT SELECT ON <table_name> to <role_name>;	**GRANT SELECT ON Studenten to MBO_read;**	Geeft SELECT privileges op table Studenten aan de role MBO_read.
GRANT <role_name> TO '<gebruikers_naam>'@'<host>';	**GRANT MBO_read TO 'User1'@'%';**	Kent de rol MBO_read toe aan de gebruiker 'User1'@'%'.
DROP ROLE <role_name>;	**DROP ROLE MBO_read;**	Verwijderd de rol MBO_read.
SET SQL_SAFE_UPDATES = 0;	**SET SQL_SAFE_UPDATES = 0;**	Zet binnen de MySQL sessie de SQL_SAVE_UPDATE modus uit om UPDATE en DELETE acties uit te kunnen voeren.
SET AUTOCOMMIT = OFF;	**SET AUTOCOMMIT = OFF;**	Zet binnen de MySQL sessie de AUTOCOMMIT modus uit voor handmatige transacties m.b.v. COMMIT of ROLLBACK.
COMMIT;	**COMMIT;**	Commiteert de openstaande transactie en is tevens het beginpunt voor de volgende transactie.
ROLLBACK;	**ROLLBACK;**	Draait de openstaande transactie terug.
SHOW PROCESSLIST;	**SHOW PROCESSLIST;**	Geeft een overzicht van de processen die op dit moment actief zijn in MySQL.
KILL <Id>;	**KILL <Id>;**	Verwijderd de sessie met Id. (Het Id nummer verkregen uit SHOW PROCESS;)
SELECT @@datadir;	**SELECT @@datadir;**	Toont de locatie(s) waar de databasebestanden staan.
SELECT @@offline_mode;	**SELECT @@offline_mode;**	Toont de waarde van de offline_mode (1 = aan, 0 = uit).

SET GLOBAL OFFLINE_MODE = 1;	**SET GLOBAL OFFLINE_MODE = 1;**	Waarde 1 zet de Offline Modus aan (gebruikerssessies worden verwijderd en gebruikers kunnen niet meer inloggen, b.v. t.b.v. onderhoud of een offline backup).
SET GLOBAL OFFLINE_MODE = 0;	**SET GLOBAL OFFLINE_MODE = 0;**	Waarde 0 zet de Offline Modus uit (gebruikers kunnen weer inloggen en wijzigingen in database maken).
SELECT <columns> FROM information_schema.TABLES GROUP BY table_schema;	**SELECT table_schema "Data Base Name", sum(data_length + index_length) / 1024 / 1024 "Data Base Size in MB", sum(data_free)/ 1024 / 1024 "Free Space in MB" FROM information_schema.TABLES GROUP BY table_schema;**	Geeft een overzicht van de gebruikte en vrije ruimte binnen de databases in MySQL.
fallocate -l <size> <file_name> fsutil file createnew <file_name> <size>	**fallocate -l 10G dummy fsutil file createnew dummy 10737418240**	Maakt en leeg bestand van 10 GB aan met de naam dummy (fallocate in Linux en fsutil in Windows).
CREATE TABLE GroteTabel AS SELECT <kolommen> FROM <table1> JOIN <table2>;	**CREATE TABLE GroteTabel AS SELECT Studenten.StudentNummer, Voornaam, Tussenvoegsel, Achternaam, ToetsCode, Cijfer FROM Studenten JOIN Cijfers;**	Maakt m.b.v. een cartesisch product een grote tabel aan voor bijvoorbeeld een performancetest.
EXPLAIN ANALYZE <query>;	**EXPLAIN ANALYZE SELECT * FROM GroteTabel WHERE Achternaam = 'Bakker';**	Geeft het execution path dat MySQL zal gebruiken om de gegevens uit de tabel te selecteren.
CREATE INDEX <index_name> ON <table_name> (column_name);	**CREATE INDEX GroteTabelAchternaam ON GroteTabel (Achternaam);**	Maakt een index GrotetabelAchternaam aan op de kolom Achternaam van de tabel GroteTabel.
ANALYZE TABLE <table_name>;	**ANALYZE TABLE Studenten;**	Verzamelt nieuwe statistieken over de gegevens in tabel Studenten.

ANALYZE TABLE <table_name> UPDATE HISTOGRAM ON <column_name>;	ANALYZE TABLE Studenten UPDATE HISTOGRAM ON Geslacht;	Genereert nieuwe histogrammen voor de statistische informatie over de gegevens in de kolom Geslacht van de tabel Studenten.
SELECT * FROM information_schema.statistics WHERE TABLE_NAME = '<table_name>';	SELECT * FROM information_schema.statistics WHERE TABLE_NAME = 'Studenten';	Geeft een overzicht van de statistische informatie over de tabel Studenten.
SELECT * FROM information_schema.column_statistics WHERE TABLE_NAME = '<table_name>';	SELECT * FROM information_schema.column_statistics WHERE TABLE_NAME = 'Studenten';	Geeft een overzicht van de statistische informatie over de kolommen van de tabel Studenten.
DELIMITER $$ CREATE PROCEDURE <procedure_name>(<paramaters>) BEGIN END$$ DELIMITER ;	DELIMITER $$ CREATE PROCEDURE spStudentPeriodeRapport(IN Student INT, IN Periode VARCHAR(10)) BEGIN SET @Periode = CONCAT('%',Periode,'%'); (SELECT v.VakNaam, ROUND(AVG(c.Cijfer),1) AS Gemiddeld FROM Cijfers AS c INNER JOIN Toetsen AS t INNER JOIN Vakken AS v ON c.ToetsCode = t.ToetsCode AND t.VakCode = v.VakCode WHERE StudentNummer = Student AND t.ToetsCode LIKE @Periode GROUP BY v.VakNaam); END$$ DELIMITER ;	Aanmaken van Stored Procedure spStudentPeriodeRapport met input parameters Student en Periode. DELIMITER $$ wordt vooraf gebruikt daar er binnen de procedure zelf ; tekens voorkomen. Het $$-teken zal hierdoor de Stored Procedure afsluiten. De commando's in de body van de procedure (tussen BEGIN en END) bevat de SQL commando's die door de Stored Procedure moeten worden uitgevoerd. Na de Stored Procedure wordt het ;-teken weer teruggezet als scheidingsteken tussen de SQL commando's.
CALL <procedure_name (parameters)>;	CALL spStudentPeriodeRapport(33,'L1P3');	Aanroep van Stored Procedure spStudentPeriodeRapport met (invoer) parameters 33 en L1P3.
DELIMITER $$ CREATE FUNCTION <function_name>() RETURNS <datatype>	DELIMITER $$ CREATE FUNCTION sfAantalStudenten() RETURNS INTEGER DETERMINISTIC	Aanmaken van Stored Function sfAantalStudenten. DELIMITER $$ wordt vooraf gebruikt daar er binnen de function zelf ; tekens voorkomen.

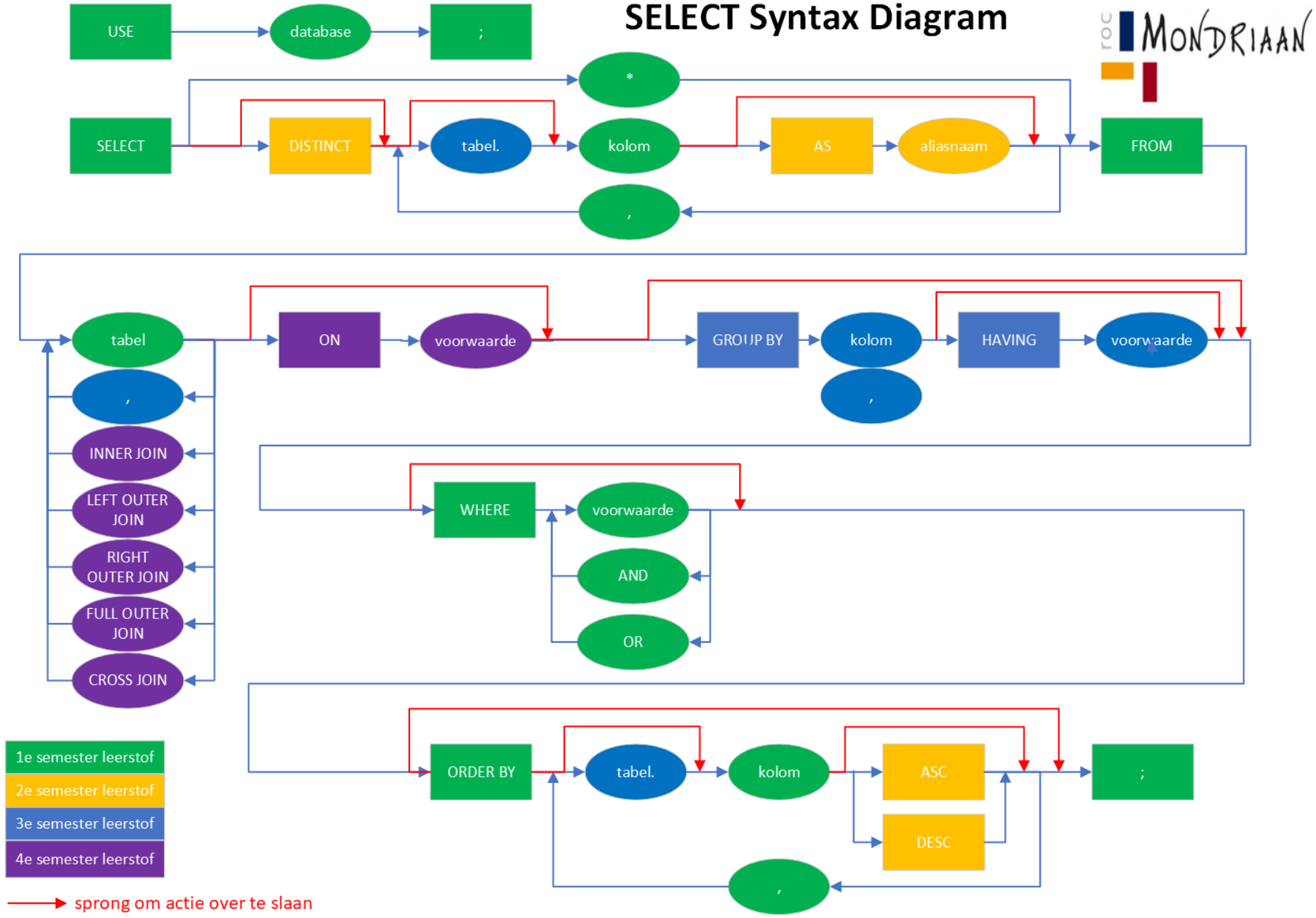
DETERMINISTIC BEGIN RETURN <value>; END$$ DELIMITER ;	BEGIN DECLARE Aantal INT; (SELECT COUNT(*) INTO Aantal FROM Studenten); RETURN Aantal; END$$ DELIMITER ;	Het $$-teken zal hierdoor de Stored Function afsluiten. De commando's in de body van de function (tussen BEGIN en END) bevat de SQL commando's die door de Stored Function moeten worden uitgevoerd. Na de Stored Function wordt het ;-teken weer teruggezet als scheidingsteken tussen de SQL commando's.
SELECT sfAantalStudenten();	SELECT sfAantalStudenten();	Aanroep van de Stored Function sfAantalStudenten.

DELIMITER $$
CREATE TRIGGER <trigger_name>
AFTER [INSERT|UPDATE|DELETE]
ON <table_name> FOR EACH ROW
BEGIN
 	
END$$
DELIMITER ;
	DELIMITER $$
	CREATE TRIGGER Salarissen_insert
	AFTER INSERT
	ON Salarissen FOR EACH ROW
	BEGIN
	insert into Salarissen_Audit_Trail(DatumTijd,ActieDoor,Actie,DocentNummer, KolomNaam, OudeWaarde, NieuweWaarde)
	values(CURRENT_TIMESTAMP,CURRENT_USER(),'INSERT',NEW.DocentNummer,'Salaris', NULL,NEW.Salaris);
	END$$
	DELIMITER ;
		Aanmaken van trigger Salarissen_insert.
		DELIMITER $$ wordt vooraf gebruikt daar er binnen de trigger zelf ; tekens voorkomen.
		Het $$-teken zal hierdoor de trigger afsluiten.
		De commando's in de body van de trigger (tussen BEGIN en END) bevat de SQL commando's die door de trigger moeten worden uitgevoerd.
		Na de trigger wordt het ;-teken weer teruggezet als scheidingsteken tussen de SQL commando's.

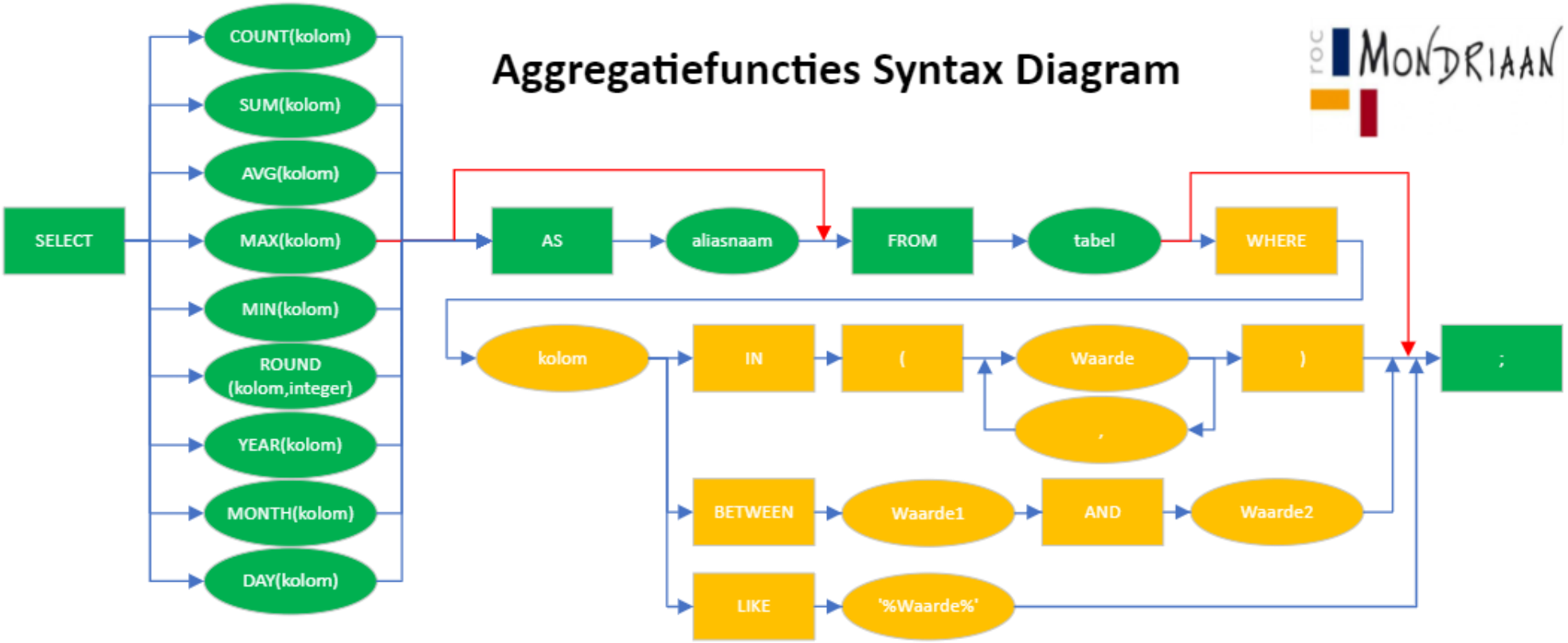
cmd -> python import mysql.connector db = mysql.connector.connect(host="localhost", user="python",password="schoolvoorict", database="Northwind") mycursor = db.cursor()	cmd -> python import mysql.connector db = mysql.connector.connect(host="localhost", user="python",password="schoolvoorict", database="Northwind") mycursor = db.cursor()	Connectie maken tussen Python en MySQL.

mycursor.execute("INSERT INTO Tabel01 VALUES ('Jan', 'Vermeer',20)")	**mycursor.execute("INSERT INTO Tabel01 VALUES ('Jan', 'Vermeer',20)")**	SQL commando in MySQL uitvoeren vanuit Python.

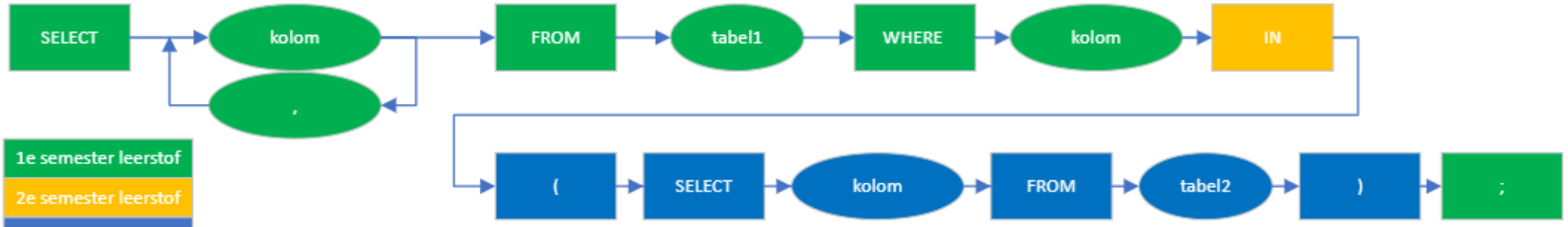
SELECT Syntax Diagram



Aggregatiefuncties Syntax Diagram



SUB-query Syntax Diagram



1e semester leerstof
2e semester leerstof
3e semester leerstof
4e semester leerstof

→ sprong om actie over te slaan