Structured query language (SQL)

Why use SQL?

SQL is a standard language that is independent of the database management systems on the market. This means that you can work with almost any kind or relationship databases if you know this language. SQL is despite its simplicity a very powerful language that let you do most of the things you need to do when working with relationship databases.

Tool

To be able to run SQL-questions with a database management system you need some kind of tool. We use MS SQL Server Management Studio. You need an account for the database server you work with, ask your lecturer for an account.

Conseption

Database:	A register that store organized data.
Table:	A structured register of data of a special kind. A table must have a unique name in a database.
Scheme:	Information about layout and characteristics of the database and tables.
Column:	A field in a table. All tables contains of one or more columns.
Data type:	Tells what kind of data that can be stored in a column. Every column in a table has its own data type, for example text or figures etc.
Row:	A post in a table.
Primary key:	One or more columns that is selected to be a unique identifier for a row in a table. A primary key is always set for each table. Two rows must not have the same value in a primary key.

Fetch, sort and filter data

To fetch data from tables and columns you use the SELECT-sentence. Sorting can be done by adding the sentence ORDER BY for the columns you use to sort by. If you would like to filter the posts you use the sentence WHERE with different operators. Some operators that can be used are =, <>, !=, <, >, <=, >=, BETWEEN, IS NULL, etc.

```
SELECT pnamn
FROM produkt
WHERE lid = 5001
ORDER BY pnamn;

SELECT pid, pnamn
FROM produkt
WHERE lid <> 5001
ORDER BY lid, pnamn;
```

More advanced data filtering can be done by writing AND, OR, IN or NOT, etc. OR and IN are almost the same, however IN is giving more easily read SQL-questions.

```
SELECT pid, pnamn
FROM produkt
WHERE lid <> 5001 AND NOT lid = 5003
ORDER BY pnamn;

SELECT pnamn, ppris
FROM produkt
WHERE lid IN (5001,5005)
ORDER BY pnamn;
```

The Operator LIKE with the joker sign % (several signs), _ (one sign) and [] (set of signs). Think of that joker signs is making the performance poorer, so don't use them more than necessary.

```
SELECT pnamn
FROM produkt
WHERE pnamn LIKE 'v%';

SELECT knamn, kkontakt
FROM Kund
WHERE kkontakt LIKE '%[CK]arlsson';
```

Calculating fields, functions, summaries and grouping of data.

In a database normally data is not saved that has been calculated. You can
I en databas lagras normalt sett inte data som kan beräknas fram. You can concatenate data (values of strings). Alias can be used to give a column a name, for example a calculated field and for calculations you use the mathematical operators -,+,* och / .

```
SELECT pid, antal, oradpris AS [á pris], antal*oradpris AS summa
FROM Orderrad
WHERE oid = 4;

SELECT (kkontakt + ' (' + knamn + ')') AS [kundkontakt (företag)]
FROM Kund;

SELECT knamn, LEFT(kkontakt,10)
FROM Kund;
```

Date can be a bit difficult. In Access you can get current date by the function NOW (). The same function in SQL Server is GETDATE(). This function can be used to generate current date for a order when it's put into the database. You can also use the function DATEPART to be able to make searches for different parts in a field with date, for example year or month:

```
SELECT * FROM [Order]
WHERE DATEPART(yy,odatum)=2006;

SELECT * FROM [Order]
WHERE odatum BETWEEN '2005-12-31' AND '2006-12-31';

SELECT * FROM [Order]
WHERE DATEPART(yy,odatum) BETWEEN 2005 AND 2007;

SELECT * FROM [Order]
WHERE (DATEPART(yy,odatum) BETWEEN 2005 AND 2006) AND
(DATEPART(mm,odatum) = 8);
```

Data can alwo be summarised in different ways with the functions AVG(), COUNT(), MIN(), MAX() and SUM().

```
SELECT SUM (antal*oradpris) AS totalt
FROM Orderrad
WHERE oid = 4;

SELECT AVG (ppris) AS Genomsnittspris, MIN(ppris) AS Minpris, MAX(ppris) AS maxpris
FROM produkt;

SELECT COUNT(*) AS [antal produkter]
FROM produkt;
```

Grouping of data and sub questions

Data can be grouped by GROUP BY to summarize subsets if needed. In the same way as WHERE can be used for filtering before grouping has been done. HAVING is used to filter rows when grouping has been done.

```
SELECT lid, COUNT(*) AS [antal produkter]
FROM produkt
GROUP BY lid;

SELECT lid, COUNT(*) AS [antal produkter]
FROM produkt
GROUP BY lid
HAVING COUNT(*) > 2;

SELECT lid, COUNT(*) AS [antal produkter]
FROM produkt
WHERE ppris >30
GROUP BY lid
HAVING COUNT(*) > 2;
```

Even sub questions can be used in another SQL question. For example we can list those customer IDs that has ordered a specific product (product ID is given). Sub questions are run first, and then the main question is run.

```
SELECT kid
FROM [order]
```

```
WHERE oid IN (SELECT oid
FROM orderrad
WHERE pid = 10001);
```

More advanced questions can also been given, but however sometimes it can be over performance. You can also use a COUNT-function to get a summary for example how many orders every customer has placed.

```
SELECT knamn, kkontakt
FROM kund
WHERE kid IN (SELECT kid
FROM [order]
WHERE oid IN (SELECT oid
FROM orderrad
WHERE pid = '10001'));

SELECT knamn, (SELECT COUNT(*)
FROM [order]
WHERE [order].kid=kund.kid) AS [antal ordrar]
FROM kund
ORDER BY knamn;
```

Connect tables and combine questions

When you separate data in different tables to be able to easier add, update and erase data it has a price – it will be more difficult to ask questions. We here look at how the more complex questions to several tables can solve this problem. Obs! The WHERE-sentence where the condition is that lid must be the same in both the primary and secondary key. If you forget to specify this you will get the Cartesian product that get all rows in the first table combined with all rows in the second table – this creates a lot of rows!

```
SELECT lnamn, pnamn, ppris
FROM leverantor, produkt
WHERE leverantor.lid=produkt.lid;

SELECT lnamn, pnamn, ppris
FROM leverantor, produkt;
```

You can also create inner connections with INNER JOIN. This means that you write a bit different from above to get “the same” answer as above. You can also connect more tables.

```
SELECT lnamn, pnamn, ppris
FROM leverantor INNER JOIN produkt
ON leverantor.lid=produkt.lid;

SELECT lnamn, pnamn, ppris, antal, odatum, knamn
FROM leverantor, produkt, [order], orderrad, kund
WHERE leverantor.lid=produkt.lid
AND orderrad.pid=produkt.pid
AND [order].oid=orderrad.oid
AND kund.kid=[order].kid;
```

Below you will find the different styles that you can choose among when using the CONVERT function to get a correct print of date:

```
SELECT lnamn, pnamn, ppris, antal, CONVERT(CHAR(24), odatum, 105)
 AS [datum], knamn
FROM leverantor, produkt, [order], orderrad, kund
WHERE leverantor.lid=produkt.lid
AND orderrad.pid=produkt.pid
AND [order].oid=orderrad.oid
AND kund.kid=[order].kid;
```

Style ID	Style Type
0 or 100	mon dd yyyy hh:miAM (or PM)
101	mm/dd/yy
102	yy.mm.dd
103	dd/mm/yy
104	dd.mm.yy
105	dd-mm-yy
106	dd mon yy
107	Mon dd, yy
108	hh:mm:ss
9 or 109	mon dd yyyy hh:mi:ss:mmmAM (or PM)
110	mm-dd-yy
111	yy/mm/dd
112	yymmdd
13 or 113	dd mon yyyy hh:mm:ss:mmm(24h)
114	hh:mi:ss:mmm(24h)
20 or 120	yyyy-mm-dd hh:mi:ss(24h)
21 or 121	yyyy-mm-dd hh:mi:ss.mmm(24h)
126	yyyy-mm-dd Thh:mm:ss.mmm(no spaces)
130	dd mon yyyy hh:mi:ss:mmmAM
131	dd/mm/yy hh:mi:ss:mmmAM

You can also do an external relationship. This means that you can list for example every product – not only those that are ordered (all products will be listed regardless if they have been ordered or not, with this external connection). If we would like to list every row in the table to the right – regardless if they have a relationship to the table to the left or not – we use RIGHT OUTER JOIN instead.

```
SELECT kund.kid, [order].oid
FROM kund LEFT OUTER JOIN [order]
ON kund.kid=[order].kid;

SELECT kund.kid, [order].oid
```

```
FROM kund RIGHT OUTER JOIN [order]
ON kund.kid=[order].kid;
```

You can also combine two questions with commands like UNION, EXCEPT and INTERSECT. These questions must have the same attribute to be able to be run together. UNION ALL makes it possible to exclude double answers. You can also just use UNION.

```
SELECT knamn,kkontakt,kmail
FROM kund
WHERE knamn ='Åhlens'
UNION ALL
SELECT knamn,kkontakt,kmail
FROM kund
WHERE kkontakt LIKE '%[CK]arlsson'
```

Insert, update and delete data

To insert values in a table you use INSERT INTO. You don't need to state all the values if you don't want to, however you can't exclude fields that are NOT NULL. If the primary key in the table is a counter, as in this case, you exclude this field (...kid INT IDENTITY(1,1) NOT NULL,... has been used when the table was created).

```
INSERT INTO kund (knamn,kadress,kpostnr,kort,kkontakt,kmail)
VALUES('Sylves El', 'Industrivägen 3','45070','Hamburgsund', 'Kenneth Johansson',
'kenneth.johansson@sylvesel.se')

INSERT INTO kund (knamn,kadress,kpostnr,kort)
VALUES('Sylves El', 'Industrivägen 3','45070','Hamburgsund')
```

If you want to add posts from one table to another you can insert a whole table at the same time by doing the following. This can also be useful if need to change the design in one table that is filled with posts (that need to be moved to a temporary table so that changes in design can be made). However this is not working so good since the reference integrity is stopping the deletion of the posts. We need to create a temporary table with the content of order and order row…

Om man vill fylla på poster ifrån en tabell till en annan kan man sätta in en hel tabell på samma gång på följande sätt. Kan även vara användbart om vi till exempel behöver ändra i designen på en tabell som är fylld av poster (som måste flyttas över i en temporär tabell så ändring av design kan ske). Nu fungerar detta emellertid dåligt eftersom referensintegriteten sätter stopp för radering av posterna, vi får även skapa temptabell med innehållet i order och orderrad...

```
-- Create temporary customer table
CREATE TABLE tempKund
(
 kid       INT NOT NULL ,
 knamn     varchar(50)       NOT NULL ,
 kadress   varchar(50)       NULL ,
```

```
  kpostnr    varchar(10)        NULL ,
  kort       varchar(50)        NULL ,
  kkontakt   varchar(50)        NULL ,
  kmail      varchar(255)       NULL
);

-- Create temporary customer row
CREATE TABLE tempOrderrad
(
  oid       INT               NOT NULL ,
  orad      INT               NOT NULL ,
  pid       INT               NOT NULL ,
  antal       INT               NOT NULL ,
  oradpris  MONEY             NOT NULL
);

-- Create temporary order
CREATE TABLE [tempOrder]
(
  oid       INT   NOT NULL ,
  odatum    SMALLDATETIME           NOT NULL ,
  kid       INT               NOT NULL
);
-- PK
ALTER TABLE tempKund WITH NOCHECK ADD CONSTRAINT PK_tempKund PRIMARY
KEY CLUSTERED (kid);
ALTER TABLE tempOrderrad WITH NOCHECK ADD CONSTRAINT PK_tempOrderrad
PRIMARY KEY CLUSTERED (oid, orad);
ALTER TABLE [tempOrder] WITH NOCHECK ADD CONSTRAINT PK_tempOrder
PRIMARY KEY CLUSTERED (oid);
-- FK
ALTER TABLE tempOrderrad ADD
CONSTRAINT FK_tempOrderrad_Order FOREIGN KEY (oid) REFERENCES [tempOrder]
(oid);
ALTER TABLE [tempOrder] ADD
CONSTRAINT FK_tempOrder_Kund FOREIGN KEY (kid) REFERENCES tempKund (kid);

INSERT INTO tempkund (kid,knamn,kadress,kpostnr,kort,kkontakt,kmail)
SELECT kid,knamn,kadress,kpostnr,kort,kkontakt,kmail FROM kund;

INSERT INTO temporder (oid,odatum,kid)
SELECT oid,odatum,kid FROM [order];

INSERT INTO temporderrad (oid,orad,pid,antal,oradpris)
SELECT oid,orad,pid,antal,oradpris FROM orderrad;

DELETE FROM orderrad;
DELETE FROM [order];
DELETE FROM kund;
```

When the posts are to be added into the table again after the change it is most likely that the numbers that have been generated with a counter aren't the same as before. A counter counts from one number and onwards all the time. A good question is how we now can get the exact same relationships as before – but with the new numbers in the keys. In this case we can start by adding all posts from the temporary table (tempkund) to the ordinary

one (kund). After that we can list the posts in kund to see which number the counter started with. You can compare this number with the one you have in tempkund. The difference is used when inserting posts from temporder to tempkund. We do in the same way for the transfer between temporder row to order row.

```
SELECT knamn,kadress,kpostnr,kort,kkontakt,kmail FROM tempkund;

INSERT INTO kund (knamn,kadress,kpostnr,kort,kkontakt,kmail)
SELECT knamn,kadress,kpostnr,kort,kkontakt,kmail FROM tempkund;

SELECT * FROM kund
SELECT * FROM tempkund

INSERT INTO [order] (odatum,kid)
SELECT temporder.odatum,kund.kid
FROM [temporder],kund,tempkund
WHERE temporder.kid=tempkund.kid AND temporder.kid=kund.kid-5;

SELECT * FROM [order] order by oid;
SELECT * FROM temporder order by oid;

INSERT INTO orderrad (oid,orad,pid,antal,oradpris)
SELECT [order].oid,tor.orad,tor.pid,tor.antal,tor.oradpris
FROM temporderrad AS tor,temporder,[order]
WHERE tor.oid=temporder.oid AND tor.oid=[order].oid-5;
```

We finally run a comparing test question to double check that everything went OK, regardless of that oid and kid can have got other values. The right customer must have the correct order as before.

```
SELECT kund.kid,[order].kid,[order].oid,orderrad.oid
FROM kund,[order],orderrad
WHERE kund.kid=[order].kid
AND [order].oid=orderrad.oid
AND knamn='Ika-Maxi';
SELECT tempkund.kid,[temporder].kid,[temporder].oid,temporderrad.oid
FROM tempkund,[temporder],temporderrad
WHERE tempkund.kid=[temporder].kid
AND [temporder].oid=temporderrad.oid
AND knamn='Ika-Maxi';
```

To update and delete data you can do this with SQL-questions as well. You can update with UPDATE and delete with DELETE. These two questions are very powerful and you need to see to that the WHERE-sentence that is used for selection of what to update is not forgotten in these kinds of questions. In the following example, two rows are updated at the same time since there are two rows that contain knamn ”Åhlens”. Think about that DELETE is removing a whole row and the order that rows are been deleted is connected to how the reference integrity is set for the table with relationships with other tables. The DELETE-sentence given as an example is not working since there are referring posts in the order table that first needs to be deleted. The order table has in its turn referring posts in order row, which needs to be deleted before order can be deleted.

```
UPDATE kund
SET kpostnr='39200', kort='Helsingborg'
WHERE kid=(SELECT kid WHERE knamn='Åhlens');

DELETE FROM kund
WHERE knamn='Åhlens';
```

Create tables and views

To create tables you use CREATE TABLE sentence. For attributes that may not be empty you state NOT NULL. You can also create different default values by using DEFAULT. A counter can be created by IDENTITY (1,1) where the first number is the starting number and the second number tell how much the number is increasing each time. GETDATE () is a function that fetch current date and time.

```
CREATE TABLE testKund
(
  kid       INT IDENTITY(1001,1) NOT NULL ,
  knamn      varchar(50)         NOT NULL ,
  kadress    varchar(50)         NULL ,
  kpostnr    varchar(10)         NULL ,
  kort       varchar(50)         NULL ,
  kkontakt   varchar(50)         NULL ,
  kmail      varchar(255)        NULL
);

CREATE TABLE [Order]
(
  oid       INT IDENTITY(1,1)   NOT NULL ,
  odatum    DATETIME            NOT NULL DEFAULT GETDATE(),
  kid       INT                 NOT NULL
);
```

To change in tables you use ALTER TABLE. For example you would like to add conditions as primary or foreign key etc. To delete an attribute or condition you use DROP COLUMN or DROP CONSTRAINT. With DROP TABLE you delete a whole table.

```
ALTER TABLE testKund
ADD testattribut varchar(10) ;

ALTER TABLE testKund WITH NOCHECK
ADD CONSTRAINT PK_testKund
PRIMARY KEY CLUSTERED (kid);

ALTER TABLE Orderrad
ADD CONSTRAINT FK_Orderrad_Order
FOREIGN KEY (oid) REFERENCES [Order] (oid),

ALTER TABLE kund
```

```
DROP COLUMN testattribut;

DROP TABLE kund;
```

To check if the number of ordered products is more than zero you add CHECK as a condition. This can be done when creating the table.

```
ALTER TABLE orderrad
ADD constraint CHECK (antal>0);

CREATE TABLE Orderrad
(
  oid       INT               NOT NULL ,
  orad      INT               NOT NULL ,
  pid       INT               NOT NULL ,
  antal         INT               NOT NULL  CHECK (antal>0),
  oradpris  MONEY             NOT NULL
);
```

Handling of views is almost the same as creating questions apart from that we write CREATE VIEW in the beginning. A view is practical to use for tailoring data for different users. Old applications then still can be used with help of a new view if the database design is altered.

```
CREATE VIEW testvy AS
SELECT knamn, kmail
FROM kund;
```

Work with stored procedures and transactions

You can create stored procedures at the database server that can help you with for example adding posts. You can run a stored procedure with EXECUTE. Often parameters like the example below are added.

```
EXECUTE nykund (’Svempas bärgning’,’Jörgen Jönsson’,’jorgen@svempas.se’);

EXECUTE nyorder (’1001’);
```

To create these procedures you can use CREATE PROCEDURE. If not a counter is used we can create a counter by building a stored procedure that do the job for us and return the new order number.

```
CREATE PROCEDURE nyorder @kid varchar(10)
AS
DECLARE @oid INTEGER
SELECT @oid=MAX(oid)
FROM [order]
SELECT @oid=@oid+1
FROM [order]
```

```
INSERT INTO [order] (oid,odatum,kid)
VALUES (@oid,GETDATE(),@kid)
RETURN@oid;
```

You can also create transaction by using COMMIT and ROLLBACK. It can be good to check if a customer already is stored as a customer before adding a new one. For example it can be good to check if it is a existing customer, since if it is, it is unnecessary to add redundant data.

```
CREATE PROCEDURE nykund
BEGIN TRANSACTION
INSERT INTO kund(kid,knamn,kmail)
VALUES('10001','Svempas bärgning','jorgen@svempas.se')
SAVE TRANSACTION startnyorder;
INSERT INTO [order](oid,odatum,kid)
VALUES(1001,'2006-10-20',10001)
IF @@ERROR <>0 ROLLBACK TRANSACTION startorder;
INSERT INTO orderrad(oid,orad,antal,pid,oradpris)
VALUES (1001,1,15,1,10.50)
IF @@ERROR <>0 ROLLBACK TRANSACTION startorder;
INSERT INTO orderrad(oid,orad,antal,pid,oradpris)
VALUES (1001,2,10,3,12.50)
IF @@ERROR <>0 ROLLBACK TRANSACTION startorder;
COMMIT TRANSACTION
```

To make a stored procedure useful for every added post we must be able to send parameters (indata) to them. If you are interested there are some more challenges to deal with if you are interested.