Structured query language (SQL)

Varför SQL?

SQL är ett standardspråk som är oberoende av databashanteringssystemen som finns på marknaden. Med andra ord kommer du kunna arbeta mot nästan alla sorters relationsdatabaser om du kan detta språk. SQL är trots sin enkelhet ett väldigt kraftfullt språk som låter dig göra det mesta som du behöver i samband med hantering av relationsdatabaser.

Verktyg

För att köra SQL-frågor mot ett databashanteringssystem krävs någon form av verktyg. Vi använder oss av MS SQL Server Management Studio. Du behöver även ett konto på på den databasserver du arbetar emot, fråga din handledare.

Begrepp

Databas:	Ett register som lagrar organiserad data.
Tabell:	En strukturerad förteckning med data av en speciell typ. Måste vara unikt namn på tabellen i en databas.
Schema:	Information om layout och egenskaper hos databas och tabeller.
Kolumn:	Ett fält i en tabell. Alla tabeller består av en eller flera kolumner.
Datatyp:	Anger vilken typ av data som kan lagras i en kolumn. Varje kolumn i en tabell har var sin datatyp, till exempel text eller siffror, etc.
Rad:	En post i en tabell.
Primärnyckel:	En eller flera kolumner som väljs ut som unik identifierare för en rad i en tabell. Anges alltid för varje tabell. Två rader får inte ha samma värde i en primärnyckel.

Hämta, sortera och filtrera data

För att hämta data från tabeller och kolumner används SELECT-satsen. Sortering kan ske med hjälp av att lägga till satsen ORDER BY för de kolumner man önskar sortering efter. Önskas filtrering av poster kan satsen WHERE användas med olika operatorer. Några operatorer som kan användas är =, <>, !=, <, >, <=, >=, BETWEEN, IS NULL, mm.

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

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

Mer avancerad datafiltrering kan göras med hjälp av AND, OR, IN och NOT, mm. OR och in är egentligen likadana, men IN ger mer lättlästa SQL-frågor.

```
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;
```

Operatorn LIKE med jokertecknet % (flera tecken), _ (ett tecken) och [] (uppsättning tecken). Tänk på att jokertecken försämrar prestandan, dvs använd dem inte i onödan.

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

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

Beräknade fält, funktioner, summering och gruppering av data

I en databas lagras normalt sett inte data som kan beräknas fram. Du kan konkatenera data (strängvärden). Alias kan användas för att namnge kolumnnamn, t ex ett beräknat fält och för beräkningar kan de matematiska operatorerna -,+,* och / användas.

```
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;
```

Datum kan vara lite bökigt. I Access kan man få aktuellt datum med hjälp av funktionen NOW(), motsvarigheten i SQL Server heter GETDATE(). Denna funktion kan t ex användas för att generera aktuellt datum för en beställning då denna läggs in i databasen. Vidare kan vi använda funktionen DATEPART för att göra sökningar för olika delar i ett datumfält, t ex år eller månad:

```
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 kan även summeras på olika sätt med funktionerna AVG(), COUNT(), MIN(), MAX() och 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;
```

Gruppering av data samt underfrågor

Data kan grupperas med GROUP BY för att exempelvis summera delmängder om så önskas. På samma sätt som WHERE kan användas för filtreringinnan gruppering har gjorts, kan HAVING användas för att filtrera rader då gruppering genomförts.

```
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;
```

Även underfrågor kan användas inne i en annan SQL fråga. T ex kan vi lista de kundid som beställt en viss produkt (produktid anges). Underfrågorna utförs först, sen körs huvudfrågan.

```
SELECT kid
FROM [order]
WHERE oid IN (SELECT oid
FROM orderrad
WHERE pid = 10001);
```

Mer avancerade frågor kan också göras, men det kan i vissa fall gå utöver prestandan. Det går även bra att baka in t ex en COUNT-funktion för att på så vis summera saker t ex hur många ordrar som varje kund har.

```
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;
```

Koppla samman tabeller och kombinera frågor

När man delar upp data i olika tabeller för att lättare kunna lägga till, uppdatera och ta bort data har det ett pris, det blir svårare att ställa frågor. Vi kikar dock här på hur de lite svårare frågorna – mot flera tabeller kan lösa detta problem. Observera WHERE-satsen villkoret att lid ska vara samma både i primär- och sekundärnyckeln. Om man glömmer specificera detta får man istället den cartesiska produkten, dvs alla rader i första tabellen kombineras med alla rader i den andra tabellen – viket ger måååånga rader.

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

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

Det går även att skapa inre kopplingar med INNER JOIN. Detta innebär att skriver lite annorlunda för att få ”samma svar” som ovan. Det går även bra att koppla ihop många fler tabeller.

```
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;
```

Nedan finner ni de olika stilar som finns att välja mellan när ni använder convertfunktionen för att få datum att visas korrekt:

```
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

Det går även att göra så kallad yttre kopplingar. Det innebär att man exempelvis kan lista samtliga produkter – inte bara de som råkar ha en beställning (oavsett om de är beställda eller inte listas alltså alla produkter med hjälp av yttre koppling). Vill vi lista alla raderna i tabellen till höger – oavsett om de har kopplingar eller inte i tabellen till vänster använder vi RIGHT OUTER JOIN istället.

```
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;
```

Du kan även kombinera två frågor med kommandon som UNION, EXCEPT och INTERSECT. Dessa frågor måste dock ha exakt samma attribut för att kunna samköras. UNION ALL gör att ej dubbletter tas bort i svaret, du kan även använda enbart UNION.

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

Infoga, uppdatera och ta bort data

För att infoga värden i tabeller använder du INSERT INTO. Du behöver ej ange alla värden om du inte vill, dock får inte fält som är NOT NULL utelämnas. Är primärnyckeln i tabellen räknare som i detta fallet, utelämnas detta fält (...kid INT IDENTITY(1,1) NOT NULL,... har använts då tabellen skapades).

```
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')
```

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...

```
-- Skapa temporär kundtabell
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
);

-- Skapa temporär orderrad
CREATE TABLE tempOrderrad
(
  oid      INT              NOT NULL ,
  orad     INT              NOT NULL ,
  pid      INT              NOT NULL ,
  antal    INT              NOT NULL ,
  oradpris MONEY            NOT NULL
```

```
);

-- Skapa temporär 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;
```

När posterna ska läggas in i tabellerna igen efter ändringen är det med allra största sannolikhet så att de nummer genererats med en räknare inte längre är likadan som innan. En räknare räknar ju ifrån ett tal och uppåt hela tiden. En bra fråga är hur vi nu ska få exakt samma kopplingar som vi hade innan – fast med de nya numren i nycklarna. I detta fall kan vi starta med att lägga till alla poster ifrån tempkund till kund. Därefter kan vi lista posterna i kund för att se vilket nummer som räknaren började räkna med. Vi kan jämföra detta nummer med det som vi har i tempkund. Mellanskillnaden använder vi vid insättning från temporder till tempkund. Vi gör på liknade sätt även för överföring ifrån temporderrad till orderrad.

```
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;
```

Vi kör slutligen en jämförande testfråga för att dubbelkolla att allt blev rätt (bortsett ifrån att oid och kid kan ha fått andra värden). Rätt kund ska hänga ihop med rätt order som innan...

```
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';
```

Uppdatering och borttagning av data kan också det göras med hjälp av SQL-frågor. Uppdatering sker med UPDATE och borttagning med DELETE. Updaterings och borttagningsfrågor är väldigt kraftfulla och det gäller att se till att den WHERE-sats som används för urval av det som ska uppdateras inte ”glöms bort” i denna typ av frågor. I exemplet nedan uppdateras till exempel 2 rader samtidigt, eftersom det finns 2 rader som innehåller knamn ”Åhlens”. Tänk även på att DELETE tar bort hela rader samt att den ordningen som rader kan tas bort hänger ihop med hur referensintegriteten ser ut för de samband som tabellen har mot andra tabeller. DELETE-satsen som ges exempel på här fungerar därför inte eftersom det finns refererande poster i order-tabellen som först måste tas bort, ordertabellen har i sin tur referande poster i orderrad som också måste bort innan order kan tas bort, osv.

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

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

Skapa tabeller och vyer

För att skapa tabeller används CREATE TABLE satser. För attribut som ej får vara tomma anger man att de ska vara NOT NULL. Det går även bra att stoppa in defaulkt-värden fär olika attribut med DEFAULT. Vidare kan räknare skapas med IDENTITY (1,1), där den första siffran är startnummer och den andra siffran är hur mycket som talet ska öka för varje ny post. GETDATE() är en funktion som hämtar dagens datum och tid.

```
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
);
```

För att ändra i tabeller används ALTER TABLE. Det t ex vara så att man vill lägga till villkor som primärnyckel eller främmande nycklar, mm. För att ta bort ett attribut eller villkor används DROP COLUMN respektive DROP CONSTRAINT. För att ta bort en hel tabell används DROP 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;
```

Det går även att kontrollera att exempelvis att antal beställda produkter är fler än noll genom att lägga ett CHECK villkor. Detta kan även låta sig göras direkt vid skapandet av tabellen.

```
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
```

```
);
```

Hantering av vyer är näst intill samma sak som att skapa frågor bortsett ifrån att vi skriver CREATE VIEW före. En vy är praktiskt för att skräddarsy data mot olika användare. Det är då möjligt att gamla applikationer kan fortsätta att fungera (med hjälp av en vy) om databasdesignen förändras.

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

Arbeta med lagrade procedurer och transaktioner

Du kan skapa lagrade procedurer på databaservern som kan hjälpa dig med t ex inläggande av poster. Du kan köra en stored procedure med EXECUTE, ofta skickas även parametrar med enligt exemplet nedan.

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

EXECUTE nyorder ('1001');
```

För att skapa dessa procedurer kan CREATE PROCEDURE användas. Om inte räknare används kan vi t ex skapa en egen räknare genom att bygga en stored procedure som utför jobbet åt oss och returnerar det nya ordernummret.

```
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;
```

Det går även att skapa transaktioner genom att använda COMMIT och ROLLBACK. T ex kan det vara skönt att kolla om en kund finns innan man lägger till kunden. Finns kunden redan är det dumt att lägga till 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
```

För att göra en stored procedure användbara för alla insättningar (inte bara en som i exemplet ovan) måste vi kunna skicka parametrar (indata) till dem. Här finns alltså lite mer utmaningar att brottas med för er som är intresserade.