SQL 2

Aggregatfunktioner

Dessa funktioner arbetar med en kolumn i en tabell och returnerar ett enskilt värde. ISO-standarden definierar fem aggregatfunktioner: SUM, AVG, COUNT, MAX och MIN. Låt oss börja med att ta en titt på COUNT. Vi vill lista hur många studenter som har högre betyg än 2.

```
SELECT COUNT(betyg) AS antal
FROM betyg
WHERE betyg > 2;
```

Resultatet av frågan visas i tabell 1.

Tabell 1

antal
2

COUNT returnerar antalet värden i en viss kolumn. COUNT(*) är en speciell användning av COUNT. Dess syfte är att räkna alla raderna i en tabell, oavsett om NULL-värden finns eller inte. Förutom COUNT(*), eliminerar varje funktion NULL-värden först och arbetar endast med de återstående icke-null-värden.

Vi vill nu lista genomsnittsbetyget för kursen med kurskoden DT022G.

```
SELECT AVG(betyg) AS medelbetyg
FROM betyg
WHERE kurskod = 'DT022G';
```

Resultatet av denna fråga kan vi se i tabell 2.

Tabell 2

medelbetyg
2.5

Funktionen AVG returnerar det aritmetiska medelvärdet av värdena i en viss kolumn.

För att lista det bästa betyget någon fått i en kurs (vilken som helst) använder vi funktionen MAX:

```
SELECT MAX(betyg) AS bästa_betyg
FROM betyg;
```

Resultatet av frågan kan ses i tabell 3.

Tabell 3

bästa_betyg
4

Du kan använda funktionerna MIN och MAX för att returnera det minsta eller högsta värdet i en specifik kolumn.

Till sist tar vi en titt på funktionen SUM. Vi vill lista summan av alla högskolepoäng för kurser där java ingår som en del av kursnamnet.

```
SELECT SUM(hp) AS java_hp
FROM kurs
WHERE namn LIKE '%java%';
```

Resultatet av frågan visas i tabell 4.

Tabell 4

java_hp
22.5

Gruppering

Vi använder GROUP BY för att gruppera resultatet av en fråga. GROUP BY ska följas av ett kolumnnamn. Data i tabellen grupperas sen efter värdena i denna kolumn. Därefter används SELECT på varje grupp och inte på varje rad. Uttrycken i SELECT-satsen måste därför vara meningsfulla att använd på gruppen, t.ex. genom att räkna raderna i gruppen eller summera värdena. Att välja ut ett enskilt värde i en rad är inte möjligt förutom värdet på den kolumn vi grupperar efter.

Vi kan skapa en fråga som listar genomsnittsbetyget för varje student.

```
SELECT portalid, avg(betyg) AS snittbetyg
FROM betyg
GROUP BY portalid
ORDER BY portalid;
```

Lägg märke till att GROUP BY skriver vi före eventuell ORDER BY. Resultatet av denna fråga kan vi se i tabell 5:

Tabell 5

portalid	snittbetyg
bean1100	2.33333333333333
kama1203	
sasv1010	3.0

För att förtydliga vad som händer kan vi dela upp det hela i olika steg. Den ursprungliga tabellen:

Tabell 6

portalid	kurskod	år	betyg
bean1100	DT022G	2012	1
bean1100	DT011G	2011	2
bean1100	DT076G	2012	4
sasv1010	DT022G	2012	4
sasv1010	DT011G	2012	2
kama1203	DT022G	2012	

Delas upp i grupper baserade på portalid (kolumnen efter GROUP BY):

Tabell 7

	portalid	kurskod	år	betyg
Gruppen "bean1100"	bean1100	DT022G	2012	1
	bean1100	DT011G	2011	2
	bean1100	DT076G	2012	4
Gruppen "sasv1010"	sasv1010	DT022G	2012	4
	sasv1010	DT011G	2012	2
Gruppen "kama1203"	kama1203	DT022G	2012	

Därefter utförs aggregatfunktionen AVG på kolumnen betyg för varje grupp:

Tabell 8

	portalid	kurskod	år	betyg
Gruppen "bean1100"	bean1100	DT022G	2012	1
	bean1100	DT011G	2011	2
	bean1100	DT076G	2012	4
			AVG=	2.333
Gruppen "sasv1010"	sasv1010	DT022G	2012	4
	sasv1010	DT011G	2012	2
			AVG=	3
Gruppen "kama1203"	kama1203	DT022G	2012	
			AVG=	

Till sist sorteras resultatet och sammanställs i resultattabellen vi kunde se i tabell 5.

Filtrera grupperingar

Uttrycket HAVING ska användas tillsammans med GROUP BY för att begränsa de grupper som visas i den slutliga resultattabellen. Även om de används på liknande sätt har HAVING och WHERE olika syften. WHERE filtrerar enskilda rader som går in i sluttabellen och HAVING filtrerar vilka grupper som går in i sluttabellen.

Vi vill återigen lista genomsnittsbetyget för varje student, men bara för studenter som har läst fler än en kurs.

```
SELECT portalid, avg(betyg) AS snittbetyg
FROM betyg
GROUP BY portalid
HAVING COUNT(portalid) > 1
ORDER BY portalid;
```

I tabell 9 kan vi se resultatet av denna fråga.

Tabell 9

portalid	snittbetyg
bean1100	2.33333333333333
sasv1010	3.0

Återigen börjar vi med att dela upp tabellen i grupper så som tabell 7 visar. Därefter utförs aggregatfunktionen AVG på kolumnen betyg samt COUNT på kolumnen portalid för varje grupp:

Tabell 10

	portalid	kurskod	år	betyg
Gruppen "bean1100"	bean1100	DT022G	2012	1
	bean1100	DT011G	2011	2
	bean1100	DT076G	2012	4
	COUNT=3		AVG=	2.333
Gruppen "sasv1010"	sasv1010	DT022G	2012	4
	sasv1010	DT011G	2012	2
	COUNT=2		AVG=	3
Gruppen "kama1203"	kama1203	DT022G	2012	
	COUNT=1		AVG=	

De grupper som inte uppfyller villkoret (eller villkoren) i HAVING-klausulen stryks helt:

Tabell 11

	portalid	kurskod	år	betyg
Gruppen "bean1100"	bean1100	DT022G	2012	1
	bean1100	DT011G	2011	2
	bean1100	DT076G	2012	4
	COUNT=3		AVG=	2.333
Gruppen "sasv1010"	sasv1010	DT022G	2012	4
	sasv1010	DT011G	2012	2
	COUNT=2		AVG=	3
Gruppen "kama1203"	~~kama1203~~	~~DT022G~~	~~2012~~	
	~~COUNT=1~~		AVG=	

Till sist sorteras resultatet och sammanställs i resultattabellen av de grupper som återstår. Resultatet kunde se i tabell 9.

Som ett sista exempel tittar vi på hur det fungerar med en WHERE tillsammans med GROUP BY. Vi vill lista genomsnittsbetyget för alla studenter, men bara för kurser som gavs under 2012 och bara för studenter som har läst fler än en kurs.

```
SELECT portalid, avg(betyg) AS snittbetyg
FROM betyg
WHERE år = '2012'
GROUP BY portalid
HAVING COUNT(portalid) > 1
ORDER BY portalid;
```

Resultatet kan vi se i tabell 12.

Tabell 12

portalid	snittbetyg
bean1100	2.5
sasv1010	3.0

Även hör börjar vi med att dela upp tabellen i grupper så som tabell 7 visar. Därefter utförs WHERE så att endast de rader som uppfyller villkoret tas med när AVG och COUNT beräknas:

Tabell 13

	portalid	kurskod	år	betyg
Gruppen "bean1100"	bean1100	DT022G	2012	1
	~~bean1100~~	~~DT011G~~	~~2011~~	~~2~~
	bean1100	DT076G	2012	4
	COUNT=2		AVG=	2.5
Gruppen "sasv1010"	sasv1010	DT022G	2012	4
	sasv1010	DT011G	2012	2
	COUNT=2		AVG=	3
Gruppen "kama1203"	kama1203	DT022G	2012	
	COUNT=1		AVG=	

De grupper som inte uppfyller villkoret (eller villkoren) i HAVING-klausulen stryks därefter helt:

Tabell 14

	portalid	kurskod	år	betyg
Gruppen "bean1100"	bean1100	DT022G	2012	1
	~~bean1100~~	~~DT011G~~	~~2011~~	~~2~~
	bean1100	DT076G	2012	4
	COUNT=2		AVG=	2.5
Gruppen "sasv1010"	sasv1010	DT022G	2012	4
	sasv1010	DT011G	2012	2
	COUNT=2		AVG=	3
Gruppen "kama1203"	~~kama1203~~	~~DT022G~~	~~2012~~	
	~~COUNT=1~~		AVG=	

Till sist sorteras resultatet och sammanställs i resultattabellen av de grupper som återstår. Resultatet kunde se i tabell 12.

Underfrågor

En underfråga (eller nästlad fråga) är en fullständig SELECT-sats inbäddad i en annan SELECT-sats. Resultatet av den inre SELECT (underfrågan) används i en yttre SELECT för att avgöra innehållet i det slutliga resultatet. En underfråga kan användas i WHERE- och HAVING-klausuler i en yttre SELECT-sats.

Vi vill lista portalid för alla studenter som har bättre genomsnittsbetyg än genomsnittet för samtliga studenter.

```
SELECT portalid
FROM betyg
GROUP BY portalid
HAVING AVG(betyg) > (
  SELECT AVG(betyg)
  FROM betyg);
```

En underfråga kan användas omedelbart efter en relationsoperator (det vill säga, =,>, < etc) i en WHERE- eller en HAVING-klausul. Att tänka på är att underfrågor som jämförs med = får endast returnera en enda rad.

Resultatet av denna fråga visas i tabell 15.

Tabell 15

portalid
sasv1010

Vi kan också använda underfrågor tillsammans med mängdoperatorerna IN och NOT IN. Dessa måste vi använda om underfrågan kan resultera i att mer än en rad hittas.

Vi vill lista portalid och namn på alla studenter som läste en kurs under år 2012.

```
SELECT portalid, förnamn, efternamn
FROM student
WHERE portalid IN (
  SELECT DISTINCT portalid
  FROM betyg
  WHERE år = '2012');
```

Resultatet kan du se i tabell 16.

Tabell 16

portalid	förnamn	efternamn
bean1100	Bertil	Andersson
kama1203	Karl	Martinsson
sasv1010	Sara	Svensson

Vi kan också använda operatörerna ANY och ALL. ANY producerar en kolumn med värden endast om villkoret uppfylls av en eller flera av de värden som ges av underfrågan. ALL producerar en kolumn med värden endast om villkoret är uppfyllt för alla värden som produceras av underfrågan.

För att förtydliga detta tittar vi på två exempel. Skapa en fråga som listar alla kurser som någon student har läst. Eller för att uttrycka det lite annorlunda. Lista alla rader i tabellen betyg som innehåller en av kurskoderna som finns i tabellen kurs.

```
SELECT DISTINCT kurskod
FROM betyg
WHERE kurskod = ANY (
  SELECT kurskod
  FROM kurs);
```

I det här fallet skulle vi i stället för = ANY kunna använda oss av IN. Resultatet ser vi i tabell 17.

Tabell 17

kurskod
DT011G
DT076G
DT022G

För att prova på användningen av ALL skriver vi en fråga som alla studenter (portalid) som har läst en kurs senare än studentens bean1100 äldsta resultat. Eller för att uttrycka det lite annorlunda. Lista alla rader i tabellen betyg som innehåller ett årtal som är högre än studentens bean1100 lägsta årtal.

```
SELECT DISTINCT portalid
FROM betyg
WHERE år > ALL (
  SELECT MIN(år)
  FROM betyg
  WHERE portalid = 'bean1100');
```

Resultatet av frågan ovan visas i tabell 18.

Tabell 18

portalid
bean1100
kama1203
sasv1010

Frågor med flera tabeller

Vi kan också skapa frågor som kombinerar rader från två tabeller när det finns motsvarande värden i en gemensam kolumn. Vi kallar det en "inre koppling" (inner join). En inner join har inte en egen relationsoperator utan består av en kombination av relationsoperatorerna: projektion, kartetisk produkt och selektion.

Syntaxen ser ut så här:

```
SELECT tabell1.kolumn1, tabell2.kolumn3
FROM tabell1, tabell2
WHERE tabell1.kolumn1 = tabell2.kolumn1;
```

Kolumn1 är namnet på den gemensamma kolumnen som tabellerna kopplas samman efter. Om kolumnerna inte är numeriska, måste de ha samma datatyp och innehålla samma typ av data, men de behöver inte nödvändigtvis ha samma namn. Oftast är det primärnyckeln i den ena tabellen som kopplas samman med främmande nyckeln i den andra tabellen.

Vi kan använda vilken av jämförelseoperatorerna som helst när vi kopplar samman två taballer: =, <, >, <=, >=, eller <>. När vi använder jämförelseoperatorn = får vi en equi-join. När vi använder någon av de andra får vi en theta-join.

Du kan använda en inner join i vilken som helst FROM-klausul. Equi-join är den vanligaste sammankopplingstypen. Den kombinerar två rader från två tabeller där det finns

motsvarande värden i kolumner som är gemensam för båda tabellerna. I den ena av tabellerna är denna kolumn primärnyckeln och i den andra tabellen är den en främmande nyckel.

För att underlätta arbetet och kanske även läsbarheten är det möjligt att ge varje tabell ett alias.

```
SELECT t1.kolumn1, t2.kolumn3
FROM tabell1 AS t1, tabell2 AS t2
WHERE t1.kolumn1 = t2.kolumn1;
```

Den grundläggande operatorn i en inner join är den kartesiska produkten. Vi kopplar då två tabeller genom att multiplicera varje rad i den första tabellen med alla rader i den andra tabellen.

Vi kan skapa en fråga som listar produkten av tabellerna student och postort:

```
SELECT *
FROM student, postort;
```

Resultatet visas i tabell 19.

Tabell 19

portalid	födelsedatum	förnamn	efternamn	adress	postnr	postnr	postort
bean1100	1970-10-10	Bertil	Andersson	Stubbvägen 1	83172	83140	Östersund
bean1100	1970-10-10	Bertil	Andersson	Stubbvägen 1	83172	83432	Brunflo
bean1100	1970-10-10	Bertil	Andersson	Stubbvägen 1	83172	89597	Skorped
bean1100	1970-10-10	Bertil	Andersson	Stubbvägen 1	83172	83172	Östersund
bean1100	1970-10-10	Bertil	Andersson	Stubbvägen 1	83172	83141	Östersund
bean1100	1970-10-10	Bertil	Andersson	Stubbvägen 1	83172	83191	Östersund
bean1100	1970-10-10	Bertil	Andersson	Stubbvägen 1	83172	80627	Gävle
bean1100	1970-10-10	Bertil	Andersson	Stubbvägen 1	83172	83145	Östersund
bean1100	1970-10-10	Bertil	Andersson	Stubbvägen 1	83172	21365	Malmö
bean1100	1970-10-10	Bertil	Andersson	Stubbvägen 1	83172	83136	Östersund
bean1100	1970-10-10	Bertil	Andersson	Stubbvägen 1	83172	11577	Stockholm
bean1100	1970-10-10	Bertil	Andersson	Stubbvägen 1	83172	62192	Visby
bean1100	1970-10-10	Bertil	Andersson	Stubbvägen 1	83172	41871	Göteborg
bean1100	1970-10-10	Bertil	Andersson	Stubbvägen 1	83172	79331	Leksand
bean1100	1970-10-10	Bertil	Andersson	Stubbvägen 1	83172	83183	Östersund
bean1100	1970-10-10	Bertil	Andersson	Stubbvägen 1	83172	37292	Kallinge
sasv1010	1985-06-06	Sara	Svensson	Blomstigen 2	83432	83140	Östersund
sasv1010	1985-06-06	Sara	Svensson	Blomstigen 2	83432	83432	Brunflo
sasv1010	1985-06-06	Sara	Svensson	Blomstigen 2	83432	89597	Skorped
sasv1010	1985-06-06	Sara	Svensson	Blomstigen 2	83432	83172	Östersund
sasv1010	1985-06-06	Sara	Svensson	Blomstigen 2	83432	83141	Östersund
sasv1010	1985-06-06	Sara	Svensson	Blomstigen 2	83432	83191	Östersund
sasv1010	1985-06-06	Sara	Svensson	Blomstigen 2	83432	80627	Gävle
sasv1010	1985-06-06	Sara	Svensson	Blomstigen 2	83432	83145	Östersund
sasv1010	1985-06-06	Sara	Svensson	Blomstigen 2	83432	21365	Malmö
sasv1010	1985-06-06	Sara	Svensson	Blomstigen 2	83432	83136	Östersund
sasv1010	1985-06-06	Sara	Svensson	Blomstigen 2	83432	11577	Stockholm
sasv1010	1985-06-06	Sara	Svensson	Blomstigen 2	83432	62192	Visby
sasv1010	1985-06-06	Sara	Svensson	Blomstigen 2	83432	41871	Göteborg
sasv1010	1985-06-06	Sara	Svensson	Blomstigen 2	83432	79331	Leksand
sasv1010	1985-06-06	Sara	Svensson	Blomstigen 2	83432	83183	Östersund
sasv1010	1985-06-06	Sara	Svensson	Blomstigen 2	83432	37292	Kallinge
kama1203	1980-12-12	Karl	Martinsson	Faktorsvägen 13	89597	83140	Östersund
kama1203	1980-12-12	Karl	Martinsson	Faktorsvägen 13	89597	83432	Brunflo
kama1203	1980-12-12	Karl	Martinsson	Faktorsvägen 13	89597	89597	Skorped
kama1203	1980-12-12	Karl	Martinsson	Faktorsvägen 13	89597	83172	Östersund
kama1203	1980-12-12	Karl	Martinsson	Faktorsvägen 13	89597	83141	Östersund

kama1203	1980-12-12	Karl	Martinsson	Faktorsvägen 13	89597	83191	Östersund
kama1203	1980-12-12	Karl	Martinsson	Faktorsvägen 13	89597	80627	Gävle
kama1203	1980-12-12	Karl	Martinsson	Faktorsvägen 13	89597	83145	Östersund
kama1203	1980-12-12	Karl	Martinsson	Faktorsvägen 13	89597	21365	Malmö
kama1203	1980-12-12	Karl	Martinsson	Faktorsvägen 13	89597	83136	Östersund
kama1203	1980-12-12	Karl	Martinsson	Faktorsvägen 13	89597	11577	Stockholm
kama1203	1980-12-12	Karl	Martinsson	Faktorsvägen 13	89597	62192	Visby
kama1203	1980-12-12	Karl	Martinsson	Faktorsvägen 13	89597	41871	Göteborg
kama1203	1980-12-12	Karl	Martinsson	Faktorsvägen 13	89597	79331	Leksand
kama1203	1980-12-12	Karl	Martinsson	Faktorsvägen 13	89597	83183	Östersund
kama1203	1980-12-12	Karl	Martinsson	Faktorsvägen 13	89597	37292	Kallinge
anan0912	1995-05-05	Anna	Andersson	Splintvägen 5	83172	83140	Östersund
anan0912	1995-05-05	Anna	Andersson	Splintvägen 5	83172	83432	Brunflo
anan0912	1995-05-05	Anna	Andersson	Splintvägen 5	83172	89597	Skorped
anan0912	1995-05-05	Anna	Andersson	Splintvägen 5	83172	83172	Östersund
anan0912	1995-05-05	Anna	Andersson	Splintvägen 5	83172	83141	Östersund
anan0912	1995-05-05	Anna	Andersson	Splintvägen 5	83172	83191	Östersund
anan0912	1995-05-05	Anna	Andersson	Splintvägen 5	83172	80627	Gävle
anan0912	1995-05-05	Anna	Andersson	Splintvägen 5	83172	83145	Östersund
anan0912	1995-05-05	Anna	Andersson	Splintvägen 5	83172	21365	Malmö
anan0912	1995-05-05	Anna	Andersson	Splintvägen 5	83172	83136	Östersund
anan0912	1995-05-05	Anna	Andersson	Splintvägen 5	83172	11577	Stockholm
anan0912	1995-05-05	Anna	Andersson	Splintvägen 5	83172	62192	Visby
anan0912	1995-05-05	Anna	Andersson	Splintvägen 5	83172	41871	Göteborg
anan0912	1995-05-05	Anna	Andersson	Splintvägen 5	83172	79331	Leksand
anan0912	1995-05-05	Anna	Andersson	Splintvägen 5	83172	83183	Östersund
anan0912	1995-05-05	Anna	Andersson	Splintvägen 5	83172	37292	Kallinge

Oftast är det inte detta vi vill uppnå. I stället vill vi hitta specifika rader i tabellerna. Till exempel som att hitta postorten till studenterna som lagras i studenttabellen. Vi kan uppnå detta om vi inkluderar en selektion. I WHERE-klausulen kan vi skriva student.postnr = postort.postnr. Resultattabellen kommer då dock att innehålla kolumnen postnr från de båda tabellerna . Vi kan utelämna en av dem genom att även inkludera en projektion i SQL-frågan. När vi gör allt detta skapar vi något som kallas för en natural equi-join.

Låt oss titta på ett exempel. Skapa en fråga som listar alla studenter med namn, postnummer och postort.

```
SELECT s.*, ort.postort
FROM student s, postort ort
WHERE s.postnr = ort.postnr
ORDER BY efternamn, förnamn, s.postnr;
```

Resultatet kan ses i tabell 20.

Tabell 20

portalid	födelsedatum	förnamn	efternamn	adress	postnr	postort
anan0912	1995-05-05	Anna	Andersson	Splintvägen 5	83172	Östersund
bean1100	1970-10-10	Bertil	Andersson	Stubbvägen 1	83172	Östersund
kama1203	1980-12-12	Karl	Martinsson	Faktorsvägen 13	89597	Skorped
sasv1010	1985-06-06	Sara	Svensson	Blomstigen 2	83432	Brunflo

Vi kan också skapa en fråga som listar hela adressen till alla studenter som heter Svensson i efternamn:

```
SELECT s.adress, s.postnr, ort.postort
FROM student s, postort ort
```

```
WHERE s.postnr = ort.postnr
  AND efternamn = 'Svensson'
ORDER BY s.postnr;
```

Resultatet av denna fråga visas i tabell 21.

Tabell 21

adress	postnr	postort
Blomstigen 2	83432	Brunflo

Vi kan även koppla samman tre eller fler tabeller med varandra. Låt säga att vi vill lista studenter, namnet på de kurser de läst, vilket år de lästa och vilket betyg de fick. Följande fråga gör detta:

```
SELECT s.portalid, s.förnamn, s.efternamn, k.namn, b.år, b.betyg
FROM student s, betyg b, kurs k
WHERE s.portalid = b.portalid
  AND b.kurskod = k.kurskod
ORDER BY efternamn, förnamn;
```

Resultatet av frågan visas i tabell 22.

Tabell 22

portalid	förnamn	efternamn	namn	år	betyg
bean1100	Bertil	Andersson	Databaser, introduktion	2012	1
bean1100	Bertil	Andersson	Operativsystem introduktionskurs	2011	2
bean1100	Bertil	Andersson	Databaser, implementering och modellering	2012	4
kama1203	Karl	Martinsson	Databaser, introduktion	2012	
sasv1010	Sara	Svensson	Databaser, introduktion	2012	4
sasv1010	Sara	Svensson	Operativsystem introduktionskurs	2012	2

Outer join

En inner join kombinerar data från två tabeller genom att bilda par av relaterade rader där de matchande kolumner i varje tabell har samma värde. Om en rad i en tabell inte är matchande kommer raden att utelämnas från resultattabellen. I ISO-standarden tillhandahåller ytterliagre ett antal operatorer som kallas för ”yttre kopplingar” (outer join). En outer join behåller rader som inte uppfyller kopplingsvillkoret.

För att förstå outer join bättre tar vi en titt på ett exempel. Lista alla postorter och de studenter som bor där. Ta även med postorter som inte har några studenter.

```
SELECT ort.postnr, ort.postort, s.efternamn, s.förnamn
FROM postort ort LEFT JOIN student s
  ON ort.postnr = s.postnr;
```

Resultatet som frågan ger visas i tabell 23.

Tabell 23

postnr	postort	efternamn	förnamn
83140	Östersund		
83432	Brunflo	Svensson	Sara
89597	Skorped	Martinsson	Karl
83172	Östersund	Andersson	Anna
83172	Östersund	Andersson	Bertil
83141	Östersund		
83191	Östersund		
80627	Gävle		
83145	Östersund		
21365	Malmö		
83136	Östersund		
11577	Stockholm		
62192	Visby		
41871	Göteborg		
79331	Leksand		
83183	Östersund		
37292	Kallinge		

En left outer join inkluderar de rader i den första (vänstra) tabellen som inte matchar någon rad från den andra (högra) tabellen (utöver att inkludera de rader som matchar). Kolumnerna från den andra tabellen fylls med NULL-värden.

Låt oss nu ha det på motsatt sätt. Lista alla studenter och de platser där de bor. Ta även med ställen där det inte finns några studenter.

```
SELECT s.efternamn, s.förnamn, ort.postnr, ort.postort
FROM student s RIGHT JOIN postort ort
  ON ort.postnr = s.postnr;
```

Resultatet av denna fråga kan vi se i tabell 24.

Tabell 24

efternamn	förnamn	postnr	postort
Andersson	Bertil	83172	Östersund
Svensson	Sara	83432	Brunflo
Martinsson	Karl	89597	Skorped
Andersson	Anna	83172	Östersund
		37292	Kallinge
		11577	Stockholm
		41871	Göteborg
		80627	Gävle
		83191	Östersund
		83141	Östersund
		83145	Östersund
		62192	Visby
		83136	Östersund
		83183	Östersund
		83140	Östersund
		21365	Malmö
		79331	Leksand

En right outer join inkluderar de rader i den andra (högra) tabellen som inte matchar någon rad från den första (vänstra) tabellen. Kolumnerna från den första tabellen fylls med NULL-värden.

Union

Unionen av två tabeller är en tabell som innehåller alla rader som är i antingen den första eller den andra eller båda tabellerna. Det viktigaste är att de två tabellerna är kompatibla med varandra. Det vill säga att de har samma struktur. Detta innebär att de två tabellerna måste innehålla samma antal kolumner och att deras motsvarande kolumner har samma datatyper och längder.

När UNION används i en fråga ger det en resultattabell från första fråga och en resultattabell från den andra frågan, som sedan slås ihop till en enda resultattabell bestående av alla rader från båda resultattabellerna med rader med dubbletter tagits bort.

Vi vill lista alla postorter från tabellerna postort och nyapostorter.

```
SELECT *
FROM nyapostorter
UNION
SELECT *
FROM postort;
```

Resultatet visas i tabell 25.

Tabell 25

postnr	postort
11577	Stockholm
21365	Malmö
37292	Kallinge
41871	Göteborg
47550	Hälsö
59055	Sturefors
62192	Visby
79331	Leksand
80627	Gävle
83136	Östersund
83140	Östersund
83141	Östersund
83145	Östersund
83172	Östersund
83183	Östersund
83191	Östersund
83432	Brunflo
89597	Skorped

Här ser vi att Stockholm endast förekommer en gång i resultattabellen även om den raden finns i båda tabellerna. Vill vi att även dubbletter ska tas med i resultattabellen kan vi använda oss av UNION ALL.

Koppla samman samma tabell

Vi kan också skriva en fråga som kopplar samman en tabell med sig själv med hjälp av alias. Vi måste använda den här tekniken om vi t.ex. vill lista de personer som läst samma kurser.

```
SELECT DISTINCT b1.portalid, b1.kurskod
FROM betyg b1, betyg b2
WHERE b1.kurskod = b2.kurskod
  AND b1.portalid <> b2.portalid
ORDER BY b1.portalid;
```

Resultatet av denna fråga kan vi se i tabell 26.

Tabell 26

portalid	kurskod
bean1100	DT011G
bean1100	DT022G
kama1203	DT022G
sasv1010	DT011G
sasv1010	DT022G

Vyer

En vy (VIEW) är en resultattabell från en SQL-fråga. Denna tabell sparas inte i databasen som vanliga tabeller utan SQL-frågan som skapade vyn körs igen varje gång man tittar på vyn. Uttryckt på ett annat sätt kan vi säga att en vy är en SQL-fråga som vi gett ett namn för att lättare kunna använda frågan i andra sammanhang.

Låt säga att vi vill skapa en fråga som listar kurskod och namn för alla kurser som handlar om databaser.

```
SELECT kurskod, namn
FROM kurs
WHERE namn LIKE '%databas%'
ORDER BY kurskod;
```

Resultatet visas i tabell 27.

Tabell 27

kurskod	kursnamn
DT022G	Databaser, introduktion
DT076G	Databaser, modellering och implementering

För att skapa en vy av denna fråga använder vi CREATE VIEW på följande sätt:

```
CREATE VIEW databaskurser
AS
SELECT kurskod, namn
FROM kurs
WHERE namn LIKE '%databas%'
ORDER BY kurskod;
```

Nu kan vi använda denna vy i andra SQL-frågor som om den vore en tabell vilken som helst.

```
SELECT *
FROM databaskurser;
```

Vilket ger oss följande resultattabell.

Tabell 28

kurskod	kursnamn
DT022G	Databaser, introduktion
DT076G	Databaser, modellering och implementering

Vi kan även använda vyn tillsammans med andra tabeller. Vi vill t.ex. lista alla studenter som läst en kurs i databaser.

```
SELECT s.portalid, s.förnamn, s.efternamn, d.kurskod, d.namn
FROM student s, betyg b, databaskurser d
WHERE s.portalid = b.portalid
  AND b.kurskod = d.kurskod;
```

Resultatet av denna fråga ser vi i tabell 29.

Tabell 29

portalid	förnamn	efternamn	kurskod	namn
bean1100	Bertil	Andersson	DT022G	Databaser, introduktion
kama1203	Karl	Martinsson	DT022G	Databaser, introduktion
sasv1010	Sara	Svensson	DT022G	Databaser, introduktion
bean1100	Bertil	Andersson	DT076G	Databaser, implementering och modellering

För att ta bort en vy skriver vi på liknande sätt som när vi tar bort en tabell, nämligen genom att använda DROP VIEW.

```
DROP VIEW databaskurser;
```

Att ta bort en vy påverkar inte på något sätt de tabeller som vyn grundar sig på. Däremot gäller det omvända. Tar vi bort en tabell som en vy grundar sig på så gör det vyn oanvändbar.