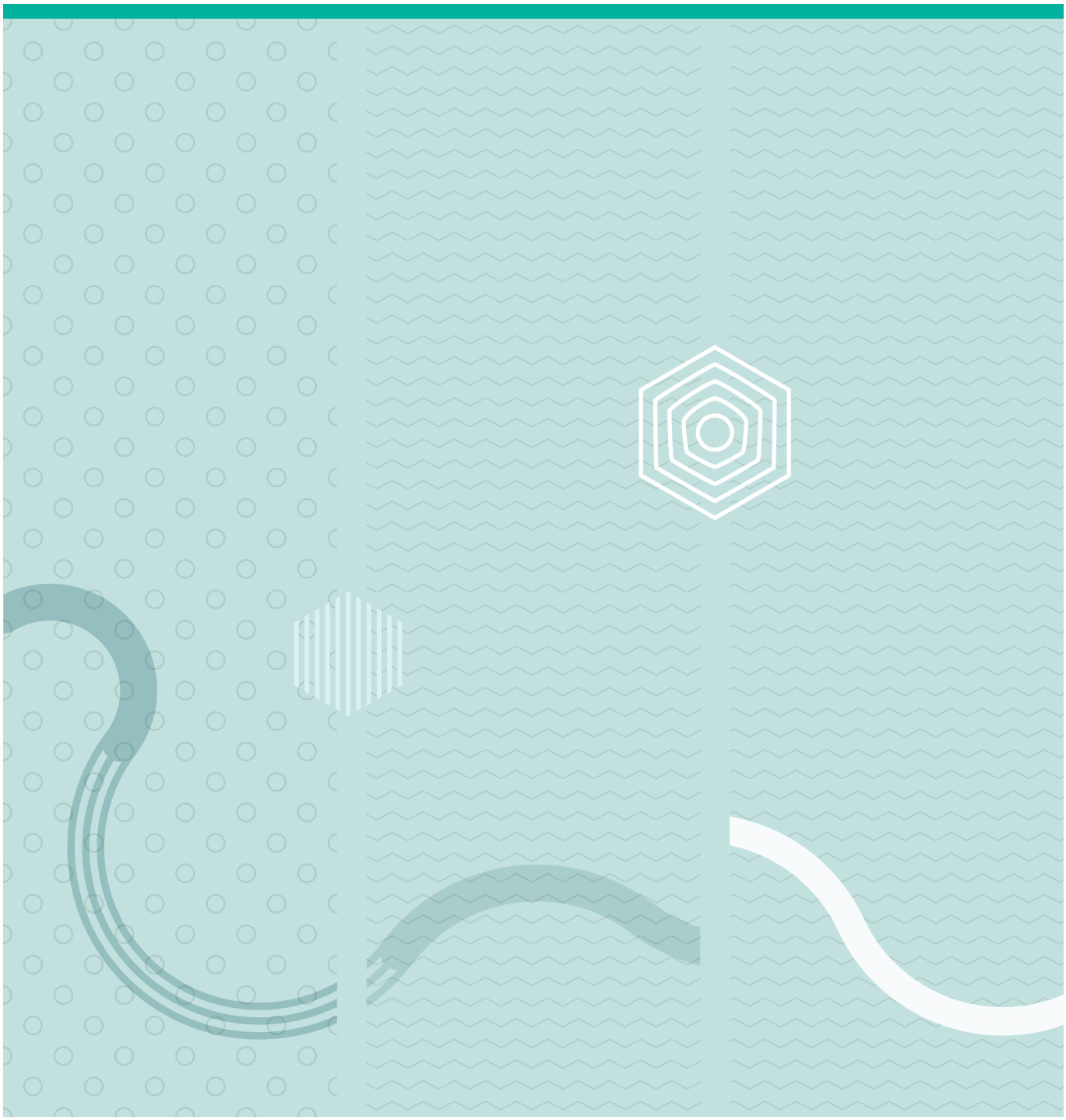
Universitetet i Sørøst-Norge
—
Skriftserien nr. 47
2020

Business Intelligence og datavarehus

En praktisk tilnærming

Trond R. Braadland



Trond R. Braadland

Business Intelligence og datavarehus

En praktisk tilnærming

Universitetet i Sørøst-Norge
Hønefoss, 2020

Skriftserien fra Universitetet i Sørøst-Norge nr. 47

ISSN: 2535-5325 (Online)
ISBN: 978-82-7860-434-2 (Online)

Forord

Dette kompendiet er skrevet for kurset BID3000- Business Intelligence og datavarehus – ved campus Ringerike i Universitetet i Sørøst-Norge. Kurset er på 7,5 studiepoeng. Det er seks hoveddeler i kurset:

- Teoretisk bakgrunn for å forstå den rollen business intelligence spiller i en virksomhet
- Teoretisk bakgrunn for oppbygning av et datavarehus
- Bruk av SQL for å bygge opp et datavarehus, fylle det med data og oppdatere det på regulær basis
- Bruk av et modelleringsverktøy for å gjøre det samme
- Bruk av dataverktøy for å presentere data som tabeller og grafer
- En kort innføring i kunstig intelligens, data mining og big data

Hovedvekten er lagt på bruk av databaser, sql og verktøy for modellering og presentasjon.

Bakgrunnen for at jeg har skrevet dette kompendiet er at ingen lærebok dekker alle disse temaene. De teoretiske delene er utførlig behandlet i bøker som Business Intelligence and Analytics: Systems for Decision Support av R. Sharda med flere, men den praktiske delen er dårlig dekket i lærebøkene. Jeg har basert de praktiske eksemplene på bruk av tilgjengelige datasett, slik at studentene kan arbeide med databaser av en viss størrelse.

Trond R. Braadland
Førstelektor
Handelshøyskolen,
Universitetet i Sørøst-Norge

Kapittel 1 Business Intelligence

Enhver virksomhet trenger informasjon om hvordan den klarer seg. Slik informasjon får den tradisjonelt gjennom regnskapet (forretningsregnskap og internregnskap), og det vil selvfølgelig være mye informasjon som utveksles mellom ansatte, både formelt og uformelt. Utviklingen av informasjonsteknologien har imidlertid gjort det mulig å skaffe informasjon langt ut over dette, og ikke minst informasjon som er helt fersk (regnskapsdata kan typisk ha en forsinkelse på et par måneder). Gjennom flere tiår har det blitt arbeidet for å lage systemer for dette. De første forsøkene kom allerede på 1960-tallet, og man kalte de systemene man søkte å lage Management Information Systems. På slutten av 1970-tallet ble rammene for mer fleksible systemer utarbeidet, og disse ble kalt Decision Support Systems (beslutningsstøttesystemer ble ganske raskt det etablerte norske begrepet). På 1980-tallet ble begrepet Executive Information Systems definert, som betegnelse på svært fleksible og brukervennlige systemer skreddersydd for topplederne. I dag brukes i stadig større grad Business Intelligence-systemer for å dekke alle disse, selv om BI egentlig er mer enn bare systemene.

Business Intelligence er blitt et viktig begrep for moderne virksomheter. Undersøkelser har vist at en av de viktigste faktorene som skiller vellykkede virksomheter fra mindre vellykkede er deres evne til kontinuerlig å drive forretningsanalyse (Hatch 2007).

Redaksjonen i Computer World Norge har foreslått å kalle BI *forretningsanalyse* på norsk. Dette begrepet dekker egentlig bedre enn BI hva dette dreier seg om.

Business Intelligence eller forretningsanalyse dekker metoder og teknologi for å fremskaffe informasjon som trengs for å ta beslutninger. D. J. Power sier det slik (http://dssresources.com/history/dsshistoryv28.html):

BI describes a set of concepts and methods to improve business decision making by using fact-based support systems. BI is sometimes used interchangeably with briefing books, report and query tools and executive information systems. Business Intelligence systems are data-driven DSS.

En annen definisjon er gitt av Turban et. al. (Turban 2011): BI's major objective is to enable interactive access (sometimes in real-time) to data, to enable manipulation of data, and to give business managers and analysts the ability to conduct appropriate analysis. By analyzing historical and current data, situation, and performances, decision makers get valuable insights that enable them to make more informed and better decisions.

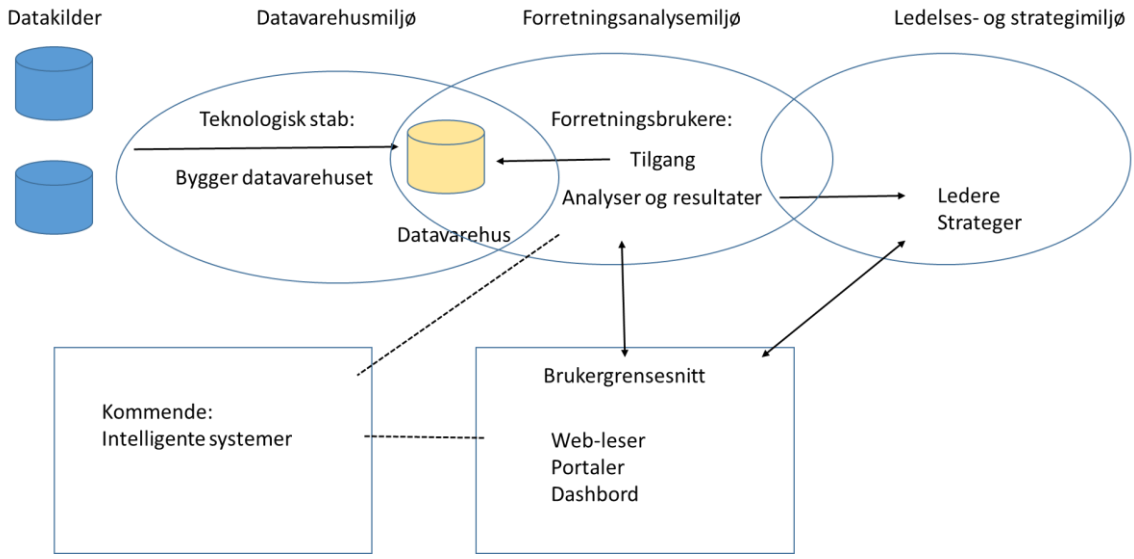
Business Intelligence går enkelt sagt ut på å bruke informasjonsteknologi til å måle og analysere hvordan virksomheten går. Dette betyr at grunnlaget for å lage et BI-system er virksomhetens strategi. Man må vite hva man skal, deretter må man finne ut hvordan man kan måle at man er på riktig vei. Endelig må man kunne hente inn de nødvendige data, lagre og analysere dem.

Teknologien bak BI omfatter alle de tidligere teknologiene bak MIS, DSS og EIS. I tillegg brukes såkalt kunstig intelligens-teknologi på deler av analysene.

I følge Turban et. al. (op.cit.) består Business Intelligence av fire hovedkomponenter:

- Et datavarehus med data fra kildesystemer
- Verktøy for forretningsanalyse
- Business Performance Management for overvåkning og analyse av ytelse
- Brukergrensesnitt (typisk et dashbord)

En fremstilling av aktører knyttet til Business Intelligence illustreres av denne figuren:



Figur 1.1 Arkitektur for BI (etter Sharda et al 2017)

Et BI-prosjekt må begynne med å bestemme hva man skal måle. Deretter må man finne ut hvor de nødvendige data skal komme fra. Det videre arbeidet vil bestå i å bygge opp et datavarehus og de nødvendige analyseverktøy og brukergrensesnitt.

1.1 Business Intelligence og programvare

I kurset skal studentene lage systemer for Business Intelligence. En lang rekke aktører leverer verktøy for dette markedet. Vi finner de virkelig store som Oracle, IBM og Microsoft, og vi finner andre store, men ikke så kjente, bedrifter som SAS Institute, Business Objects og Teradata.

Vi skal dels bruke verktøy av typen åpen kildekode, dels tilgjengelig programvare fra Microsoft og Visual Paradigm. Åpen kildekode-verktøyet vi skal bruke er av typen Community Edition, som er gratis. For alle Community Edition programvare finnes også lisensutgaver, som typisk har mer funksjonalitet, mindre feil og dessuten har support. De CE-utgavene vi skal bruke virker imidlertid helt greit for våre formål.

MySQL (www.mysql.com) er et åpen kildekode databasesystem. Det finnes i to utgaver: en Community-utgave som er gratis for ikke-kommersiell bruk, og en Enterprise-utgave som man må betale for. En forskjell på disse to er at Enterprise-utgaven oppdateres ofte, og man må regne med at den har færre feil enn Community-utgaven. For undervisningsbruk skulle ikke dette spille noen rolle. MySQL er i dag mye brukt som databasesystem for store kommersielle nettapplikasjoner, da helst i kombinasjon med PhP. MySQL er et av

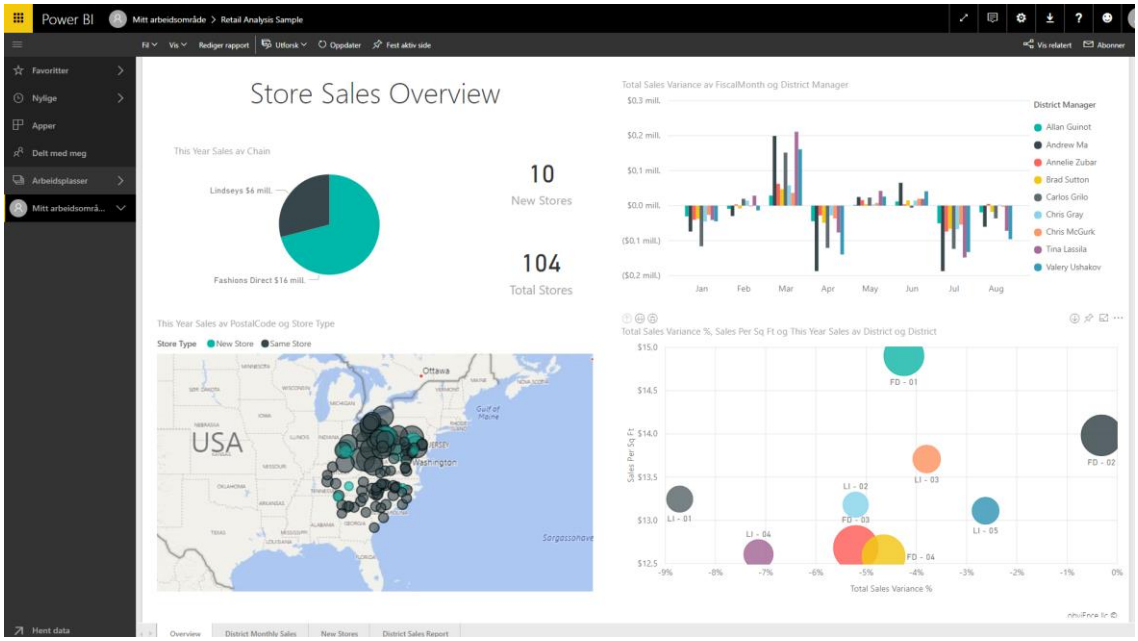
elementene innen det ettertraktede kompetanseområdet som kalles LAMP – Linux, Apache, MySQL, PhP.

Pentaho (www.pentaho.org) er en stor aktør på programvare for å utvikle BI-løsninger. Disse verktøyene spenner fra modelleringsverktøy for å spesifisere ETL (Spoon), til verktøy for OLAP (Mondrian) og Data Mining (Weka). Vi skal bruke ETL-verktøyet Spoon.

Microsofts Excel og Power BI skal vi bruke som analyseverktøy.

1.2 Demonstrasjon av et BI-system

Vi skal se på noen eksempler på det som er front-end delen av Business Intelligence-systemer, altså det brukerne møter. Eksemplene er delvis hentet fra et system som heter Pentaho BI Server Community Edition, delvis fra Microsoft Power BI. Vi starter med å vise et typisk dashbord, i dette tilfellet fra Power BI:



Figur 1.2 Dashbord fra Microsoft Power BI (https://powerbi.microsoft.com/en-us/)

Dashbordet viser forskjellige visualiseringer av analyser av data. Brukeren kan så velge en av dem og dermed bli brakt videre til en mer detaljert visualisering.

Vi skal bruke denne demo-applikasjonen til å se på hvilke analysemuligheter brukerne av Business Intelligence systemer tilbys. Typisk kan vi dele analysetilbudene i fire kategorier:

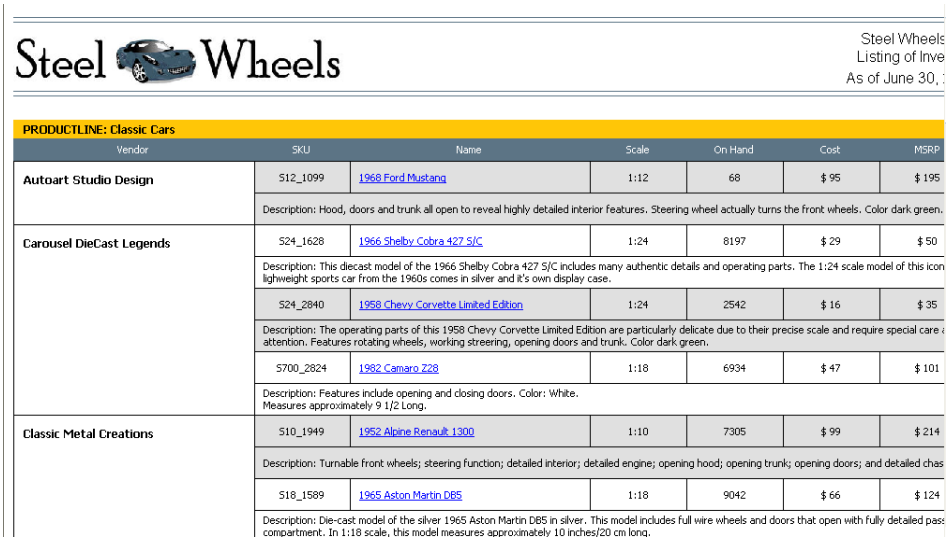
- Rapporter
- Dashbord
- Online Analytical Processing (OLAP)
- Data Mining (datagruvedrift)

Pentaho-demoen gir oss eksempler på de tre første kategoriene.

Datagrunnlaget er en database kalt SteelWheels eller Classic Cars. Det er en database som har vært brukt i mange år som grunnlag for opplæring i datavarehus. Jeg har selv vært på et datavarehus-kurs hos Oracle Norge der denne databasen ble brukt.

Databasen inneholder salgsdata for et firma som selger modeller av biler, båter, fly med mer. Dette er kildesystemet som brukes for å lage et datavarehus. I datavarehuset er dataene strukturert slik at de er optimale for spørringer (som kjent er et transaksjonsprosesseringssystem optimalisert for transaksjoner). I løpet av kurset skal vi lage datavarehus ut fra TPS'er.

Vi starter med å presentere en rapport laget på grunnlag av datavarehuset:



Steel Wheels

Steel Wheels
Listing of Inve
As of June 30,

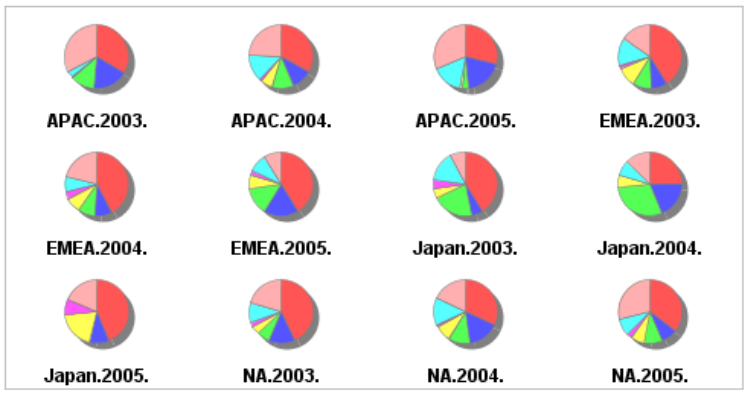
PRODUCTLINE: Classic Cars

Vendor	SKU	Name	Scale	On Hand	Cost	MSRP
Autoart Studio Design	S12_1099	1968 Ford Mustang	1:12	68	$ 95	$ 195
	Description: Hood, doors and trunk all open to reveal highly detailed interior features. Steering wheel actually turns the front wheels. Color dark green.					
Carousel DieCast Legends	S24_1628	1966 Shelby Cobra 427 S/C	1:24	8197	$ 29	$ 50
	Description: This diecast model of the 1966 Shelby Cobra 427 S/C includes many authentic details and operating parts. The 1:24 scale model of this icon lighweight sports car from the 1960s comes in silver and it's own display case.					
	S24_2840	1958 Chevy Corvette Limited Edition	1:24	2542	$ 16	$ 35
	Description: The operating parts of this 1958 Chevy Corvette Limited Edition are particularly delicate due to their precise scale and require special care attention. Features rotating wheels, working streering, opening doors and trunk. Color dark green.					
	S700_2824	1982 Camaro Z28	1:18	6934	$ 47	$ 101
	Description: Features include opening and closing doors. Color: White. Measures approximately 9 1/2 Long.					
Classic Metal Creations	S10_1949	1952 Alpine Renault 1300	1:10	7305	$ 99	$ 214
	Description: Turnable front wheels; steering function; detailed interior; detailed engine; opening hood; opening trunk; opening doors; and detailed chas					
	S18_1589	1965 Aston Martin DB5	1:18	9042	$ 66	$ 124
	Description: Die-cast model of the silver 1965 Aston Martin DB5 in silver. This model includes full wire wheels and doors that open with fully detailed pass compartment. In 1:18 scale, this model measures approximately 10 inches/20 cm long.					

Figur 1.3 Rapport fra Pentaho BI Server Community Edition (https://community.hitachivantara.com/s/pentaho)

Grafiske presentasjoner er mye brukt for å visualisere data. Nedenfor er vist bruk av kakediagrammer. Disse virker slik av hvis vi klikker på et diagram, blir vi sendt videre til en *krysstabell*. Denne byr på det vi kaller *drilling*:

Drill Down to Pivot Table



Slicer: Indikator=Sales

Classic Cars. Motorcycles. Planes. Ships. Trains. Trucks and Buses. Vintage Cars.

	Markets											
	+APAC			+EMEA			+Japan			+NA		
	Time			Time			Time			Time		
Product	+2003	+2004	+2005	+2003	+2004	+2005	+2003	+2004	+2005	+2003	+2004	+2005
+Classic Cars	115 011	199 372	97 574	691 273	1 015 790	384 538	120 696	42 071	18 835	587 428	581 043	237 791
+Motorcycles	60 789	63 159	65 870	141 836	204 042	161 260	16 485	31 959	4 176	178 109	291 421	55 020
+Planes	42 663	67 681	11 082	154 519	209 128	128 008	60 556	49 177		90 016	202 942	60 985
+Ships		35 323	3 070	172 428	186 992	67 845	14 156	10 453	8 407	58 238	142 904	48 856
+Trains	1 681	8 226		29 538	90 973	17 995	13 279		3 524	28 304	25 551	15 398
+Trucks and Buses	11 298	80 634	53 735	228 699	185 421	86 859	44 498	13 349		135 936	252 572	61 281
+Vintage Cars	111 639	147 212	105 688	263 695	504 062	83 324	22 888	21 470	7 979	281 727	324 815	191 727

Figur 1.4 Grafikk og krysstabell med drilling fra Pentaho BI Server (ibid.)

Vi kan foreta drilling ved å klikke på + foran de forskjellige produktkategoriene, slik vi er vant med fra katalogtreet i Windows utforskeren. Pivottabellen gir oss nå nye tall for objektene i kategorien, her ”Classic Cars”:

	Markets											
	+APAC			+EMEA			+Japan			+NA		
	Time			Time			Time			Time		
Product	+2003	+2004	+2005	+2003	+2004	+2005	+2003	+2004	+2005	+2003	+2004	+2005
−Classic Cars	115 011	199 372	97 574	691 273	1 015 790	384 538	120 696	42 071	18 835	587 428	581 043	237 791
+Autoart Studio Design	24 647	44 301	40 608	124 844	186 058	69 478	17 939	14 200		100 561	139 490	37 517
+Carousel DieCast Legends	16 420	46 818	15 177	160 735	175 029	40 761	21 999	6 135	6 028	59 541	135 449	65 705
+Classic Metal Creations	48 714	39 945	32 259	148 995	263 394	90 209	37 545	17 504	1 406	125 059	153 603	65 034
+Exoto Designs	42 225	54 527	31 170	131 247	202 488	64 877	15 297	3 676	7 738	115 039	157 809	53 762
+Gearbox Collectibles	40 045	56 384	24 938	138 703	232 697	76 927	22 560	17 019		131 704	129 519	42 426
+Highway 66 Mini Classics	26 142	35 495	20 187	107 010	145 865	93 494	16 027	21 774		91 951	156 465	33 550
+Min Lin Diecast	20 031	41 506	15 442	124 317	162 047	70 363	12 531	17 934	7 032	109 562	133 179	50 285
+Motor City Art Classics	25 519	65 203	40 042	141 014	157 157	52 825	18 884	7 331		88 195	133 417	79 691
+Red Start Diecast	19 374	51 779	16 851	122 495	131 053	72 069	13 139	19 779	554	93 476	152 714	37 310
+Second Gear Diecast	12 640	47 399	17 270	111 746	215 669	50 993	44 034	12 227	5 941	122 702	148 284	68 947
+Studio M Art Models	27 519	27 764	31 054	68 498	130 710	42 134	16 944	10 493	4 497	70 686	85 051	51 985
+Unimax Art Galleries	20 136	42 224	21 001	173 776	196 657	128 581	25 260	6 759	3 863	128 206	190 542	34 567
+Welly Diecast Productions	19 670	48 262	31 019	128 607	197 583	77 117	30 399	13 648	5 862	123 076	105 725	50 279
+Motorcycles	60 789	63 159	65 870	141 836	204 042	161 260	16 485	31 959	4 176	178 109	291 421	55 020
+Planes	42 663	67 681	11 082	154 519	209 128	128 008	60 556	49 177		90 016	202 942	60 985
+Ships		35 323	3 070	172 428	186 992	67 845	14 156	10 453	8 407	58 238	142 904	48 856
+Trains	1 681	8 226		29 538	90 973	17 995	13 279		3 524	28 304	25 551	15 398
+Trucks and Buses	11 298	80 634	53 735	228 699	185 421	86 859	44 498	13 349		135 936	252 572	61 281
+Vintage Cars	111 639	147 212	105 688	263 695	504 062	83 324	22 888	21 470	7 979	281 727	324 815	191 727

Figur 1.5 Krysstabell med drilling på e produktgruppe (ibid.)

Som vi ser står det fortsatt + foran underkategoriene under Classic Cars. Vi trykker på + for "Classic Metal Creations", og driller enda lenger ned:

	Markets											
	+APAC			+EMEA			+Japan			+NA		
	Time			Time			Time			Time		
Product	+2003	+2004	+2005	+2003	+2004	+2005	+2003	+2004	+2005	+2003	+2004	+2005
−Classic Cars	115 011	199 372	97 574	691 273	1 015 790	384 538	120 696	42 071	18 835	587 428	581 043	237 791
+Autoart Studio Design	24 647	44 301	40 608	124 844	186 058	69 478	17 939	14 200		100 561	139 490	37 517
+Carousel DieCast Legends	16 420	46 818	15 177	160 735	175 029	40 761	21 999	6 135	6 028	59 541	135 449	65 705
−Classic Metal Creations	48 714	39 945	32 259	148 995	263 394	90 209	37 545	17 504	1 406	125 059	153 603	65 034
1928 British Royal Navy Airplane	10 514	6 143	4 329	10 454	19 050	11 094	7 629	3 723		9 935	10 464	3 659
1938 Cadillac V-16 Presidential Limousine	2 432	1 846	5 445	6 229	18 378	1 845				6 110	4 366	2 685
1949 Jaguar XK 120	4 472	5 957	3 934	10 804	22 472	1 350				14 988	8 373	11 289
1952 Alpine Renault 1300	8 015	8 850	2 417	27 233	25 324	12 001	10 993	7 681		26 672	44 978	16 910
1954 Greyhound Scenicruiser		1 753		12 186	16 907	5 369	2 208			5 345	16 883	1 329
1956 Porsche 356A Coupe		5 014		27 896	39 157	25 670	5 449			20 330	10 349	6 763
1957 Corvette Convertible	10 797	6 276	10 762	13 386	42 652	4 919	5 565	6 101		13 769	16 026	6 863
1961 Chevrolet Impala	6 635		5 374	6 972	27 278	10 049	2 130			5 556	16 922	2 473
1962 City of Detroit Streetcar		1 169		5 934	27 831	11 259	3 571		1 406	9 099	9 138	3 256
1965 Aston Martin DB5	5 849	2 937		27 901	24 345	6 653				13 256	16 105	9 807
+Exoto Designs	42 225	54 527	31 170	131 247	202 488	64 877	15 297	3 676	7 738	115 039	157 809	53 762
+Gearbox Collectibles	40 045	56 384	24 938	138 703	232 697	76 927	22 560	17 019		131 704	129 519	42 426
+Highway 66 Mini Classics	26 142	35 495	20 187	107 010	145 865	93 494	16 027	21 774		91 951	156 465	33 550
+Min Lin Diecast	20 031	41 506	15 442	124 317	162 047	70 363	12 531	17 934	7 032	109 562	133 179	50 285
+Motor City Art Classics	25 519	65 203	40 042	141 014	157 157	52 825	18 884	7 331		88 195	133 417	79 691
+Red Start Diecast	19 374	51 779	16 851	122 495	131 053	72 069	13 139	19 779	554	93 476	152 714	37 310
+Second Gear Diecast	12 640	47 399	17 270	111 746	215 669	50 993	44 034	12 227	5 941	122 702	148 284	68 947
+Studio M Art Models	27 519	27 764	31 054	68 498	130 710	42 134	16 944	10 493	4 497	70 686	85 051	51 985
+Unimax Art Galleries	20 136	42 224	21 001	173 776	196 657	128 581	25 260	6 759	3 863	128 206	190 542	34 567
+Welly Diecast Productions	19 670	48 262	31 019	128 607	197 583	77 117	30 399	13 648	5 862	123 076	105 725	50 279
+Motorcycles	60 789	63 159	65 870	141 836	204 042	161 260	16 485	31 959	4 176	178 109	291 421	55 020
+Planes	42 663	67 681	11 082	154 519	209 128	128 008	60 556	49 177		90 016	202 942	60 985

Figur 1.6 Videre drilling (ibid.)

Vi er nå nede på enkeltprodukt. Vi kan imidlertid også drille på kategoriene på X-aksen i pivottabellen. Som vi ser står de + også foran disse. Et klikk på + foran EMEA (Europeiske land) gir oss denne tabellen (rettere sagt litt av den):

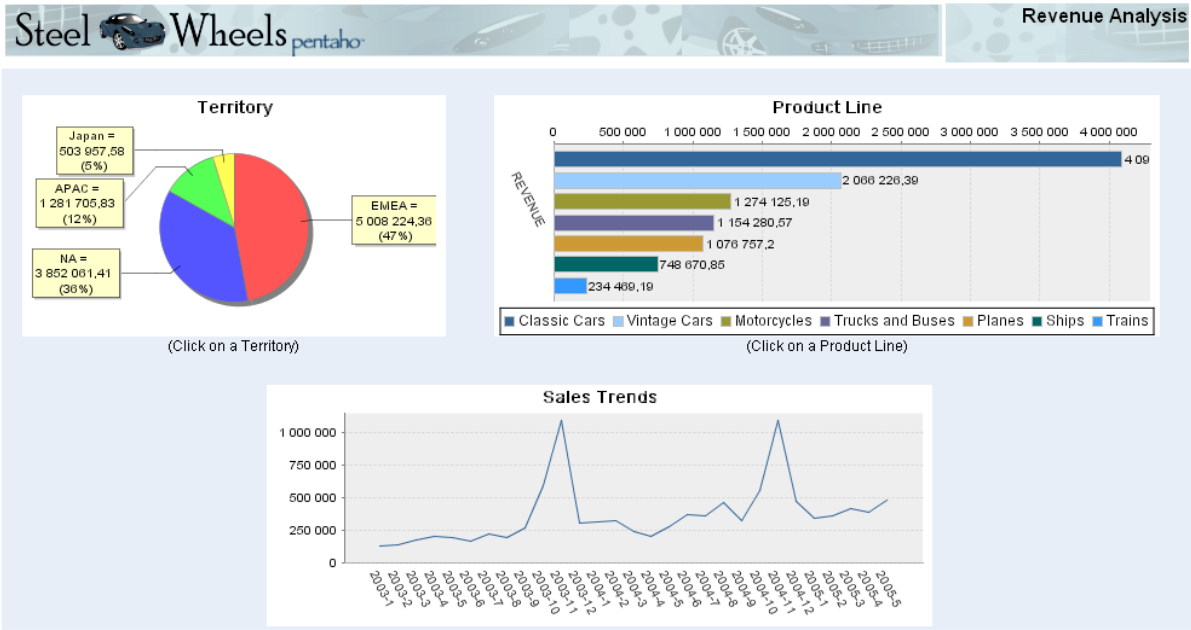
	Markets																	
	+APAC			-EMEA			+Austria			+Belgium			+Denmark			+Finland		
	Time			Time			Time			Time			Time			Time		
Product	+2003	+2004	+2005	+2003	+2004	+2005	+2003	+2004	+2005	+2003	+2004	+2005	+2003	+2004	+2005	+2003	+2004	+2005
+Classic Cars	115 011	199 372	97 574	691 273	1 015 790	384 538	26 675	15 309	59 475		3 509	16 628	60 786	70 384	26 013	33 118	54 249	66 186
+Motorcycles	60 789	63 159	65 870	141 836	204 042	161 260		26 048									32 210	15 657
+Planes	42 663	67 681	11 082	154 519	209 128	128 008	14 216	3 644			5 625			7 586		23 113		11 262
+Ships		35 323	3 070	172 428	186 992	67 845	9 025				31 708		20 452	18 245		6 408		23 401
+Trains	1 681	8 226		29 538	90 973	17 995				1 711	7 306		4 330	7 146			5 117	
+Trucks and Buses	11 298	80 634	53 735	228 699	185 421	86 859	20 473							9 589		40 479		
+Vintage Cars	111 639	147 212	105 688	263 695	504 062	83 324	11 729	6 693	8 775	1 637	31 876	8 412	13 625	7 481		8 037		10 346

Slicer: [Indikator=Sales]

Figur 1.7 Drilling på salgsområde (ibid.)

Vi kan selvsagt kombinere drilling på de to aksene.

Vi kan nå gå tilbake til hovedvalgene for Steel Wheels, og velger Dashboards. Vi kan da få dette dashbordet:

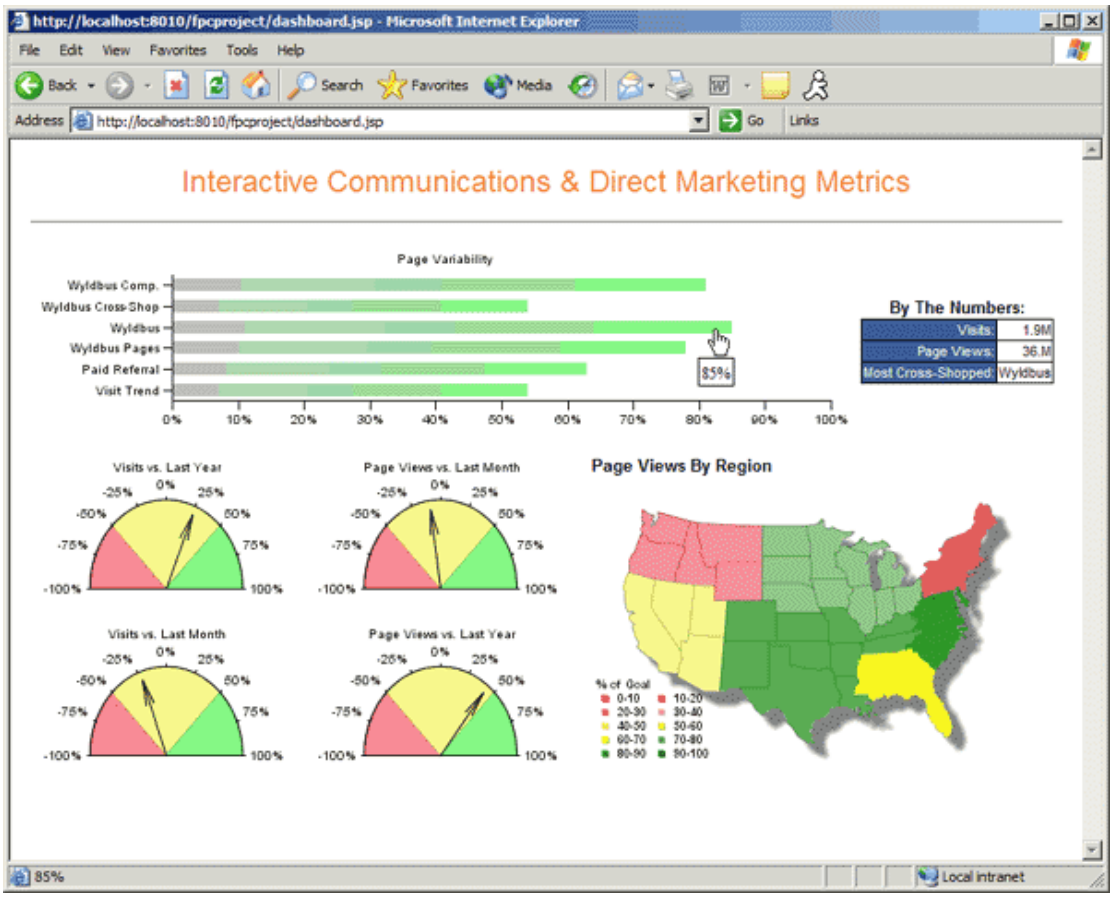


Figur 1.8 Dashbord fra Pentaho BI Server (ibid.)

Det er populært å bruke kart i forbindelse med dashbord. Her i landet er det for eksempel mange virksomheter som kjøper kart fra Statens kartverk i sine applikasjoner. Pentaho har lagt opp til at man kan bruke kart fra Google Map. Her kan man velge hvordan verdier skal vises på kartet. Det er vanlig å bruke rødt for verdier som ligger under et kritisk nivå, gult eller oransje for mellomverdier og grønt for verdier som er OK. Dette defineres for eksempel på denne måten:

Figur 1.9 Bruk av kart i BI (ibid.)

Det er også vanlig å lage dashbord der forskjellige visualiseringer kombineres med kart. Nedenfor er vist et eksempel på et slikt, hentet fra www.visualmining.com :



Figur 1.10 Visualisering med tabeller, «målere» og kart (ibid.)

Kapittel 2 Grunnlaget for det hele: mål og strategi

Strategi er opprinnelig et militært begrep (noe det fortsatt er), som er blitt tatt i bruk i ledelsesfag. I Quinn (1988, pp. 1 - 2) er det (greske) militære begrepet forklart slik: "Initially *strategos* referred to a role (a general in command of an army). Later it came to mean "the art of the general", which is to say the psychological and behavioral skills with which he occupied the role. By the time of Perikles (350 BC) it came to mean managerial skill (administration, leadership, oration, power). And by Alexander's time (330 BC) it referred to the skill of employing forces to overcome opposition and to create a unified system of global governance". I dag brukes strategibegrepet i mange sammenhenger, ikke minst i organisasjoner. Det å utarbeide den riktige strategien, og så følge den, er essensielt for de fleste organisasjoner. I dette kapittelet skal vi se nærmere på strategiarbeid og den rolle informasjonsteknologien spiller i den sammenheng.

2.1 Grunnleggende begreper

Blant forfattere innen strategifaget brukes forskjellige definisjoner på moderne forretningsstrategi. Quinn (1988, p. 3) definerer strategi slik: "A strategy is the *pattern* or *plan* that *integrates* an organization's *major* goals, policies, and action sequences into a *cohesive* whole. A well-formulated strategy helps to *marshal* and *allocate* an organization's resources into a *unique and viable posture* based on its relative *internal competencies* and *shortcomings*, anticipated *changes in the environment*, and contingent moves by *intelligent opponents*" (forfatterens uthevelser).

Strategien er altså planer som, med utgangspunkt i organisasjonens mål, skal resultere i aktiviteter for allokering av ressurser. Dette er igjen basert på organisasjonens kompetanse, forventede endringer i omgivelsene og konkurrentenes handlinger. Denne definisjonen rommer essensen i strategibegrepet slik det vanligvis brukes.

Organisasjonens mål er altså overordnet strategien. Målene bør formuleres slik at man faktisk kan måle om de er oppnådd. Eksempler på mål kan være "Vi skal ha minst 30 % markedsandel innen vår bransje i Norge" eller "Vi skal ha minst 10 % fortjeneste av vår omsetning". Slike mål er ikke bare utgangspunkt for strategien, men skal også brukes til å styre organisasjonen etter. For mange organisasjoner er målene mer på "festtalenivå", og dermed lite egnet til å styre etter.

Taktikk er et annet begrep som henger nøye sammen med strategi. Forskjellen på taktikk og strategi ligger i perspektivet: strategi er rettet mot å oppnå overordnede og langsiktige mål, mens taktikk er mer kortsiktig og rettet mot å oppnå delmål. Vi ser dette best i den militære definisjonen, her uttrykt av militærteoretikeren von Clausewitz (Roos 2002):

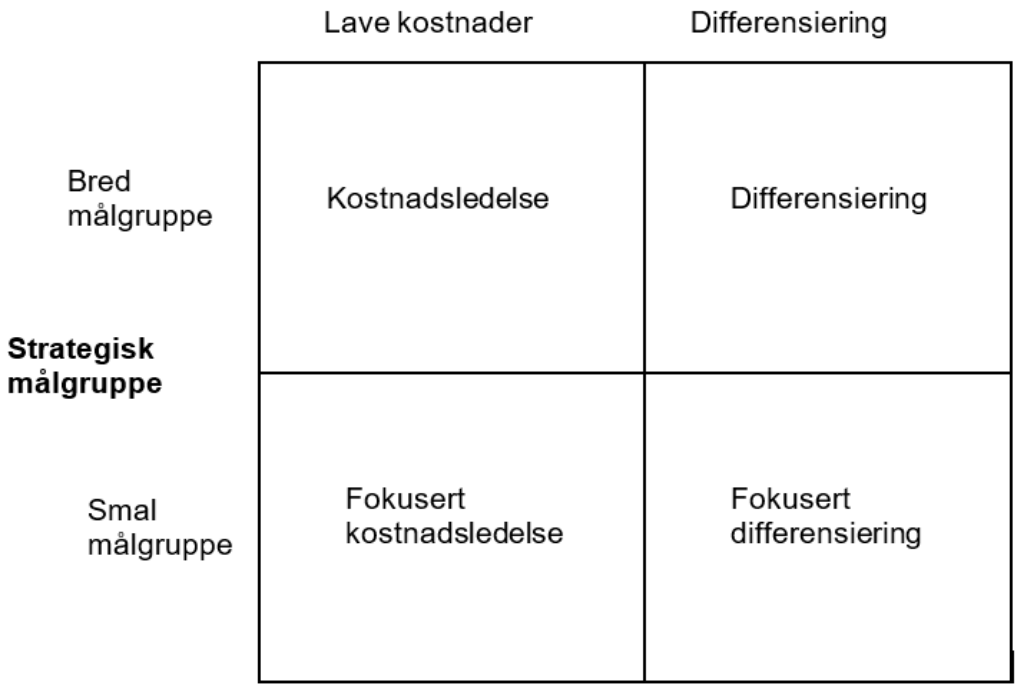
Dersom krig var en enkelt handling, ville det ikke være noe behov for en oppdeling i delaktiviteter. Imidlertid er krigshandlinger sammensatt av et større eller mindre antall delaktiviteter som er komplette i seg selv. Dette kalles slag (…). Ut fra dette kan to totalt forskjellige arbeidsområder identifiseres. Det ene er årsaken til og gjennomføringen av de enkelte slagene, og det andre er kombinasjonen av dem sett I lyset av det overordnede målet for krigen. Det første blir kalt taktikk og det andre strategi.

De to begrepene glir over i hverandre avhengige av perspektivet som brukes. Det som for toppledelsen i en virksomhet kan bli sett på som taktikk, kan for en avdelingsleder være strategi.

En god strategi regnes i dag for å være nøkkelen til suksess for en virksomhet. Samtidig som virksomheten må ha en strategi (og følge den), må den også være i stand til å legge om strategien dersom ytre forhold skulle tilsi det. Strategiarbeidet vil alltid resultere i planer for fremtiden. Mens bedrifter på 1950-tallet kunne legge planer for 50 år fremover, har den typiske strategiplan i dag en gyldighet på 3 til 5 år. Verden omkring virksomheten endrer seg for fort til at det gir mening å planlegge for lengre tid, samtidig som planene også må ha en viss gyldighet for at de skal kunne brukes.

2.2 Forretningsstrategier

Utgangspunktet for en virksomhet er at det er omgivelser man må tilpasse seg. For en bedrift, som lever av å selge varer eller tjenester, er det flere forhold i omgivelsene som må tas i betraktning ved utforming av strategi. Samtidig må virksomhetens interne organisering tilpasses de oppgaver den skal utføre. Overført fra militær til bedriftsøkonomisk tenkning, kan vi si at strategi "innebærer en rekke planlagte tiltak som er fastsatt på forhånd, og som blir vedtatt for å oppnå et bestemt mål" (Roos 2002).



Figur 2.1 Forretningsstrategier (etter Porter 1980)

Før man kan begynne å planlegge, må man imidlertid ta noen beslutninger angående den overordnede strategien. Spørsmålet her er hvordan man skal tilnærme seg markedet. Harvard-professoren Michael Porter har definert tre forskjellige *forretningsstrategier* eller *generiske strategier*, som er karakterisert ved *strategisk målgruppe* og *konkurransefortrinn* (Porter 1980). Den strategiske målgruppen kan være bred eller smal, mens

konkurransefortrinnet kan være basert på lave kostnader eller på differensiering (kundetilpasning). Dette kan illustreres med matrisen i figur 2.1

2.2.1 Kostnadsledelse

Kostnadsledelse betyr ganske enkelt at man konkurrerer på pris, dvs. at man tilbyr produktene billigere enn konkurrentene. Implisitt ligger det i dette at man opererer med standardiserte produkter i et bredt marked. Sagt på en annen måte: man prøver å selge billigst mulig til flest mulig. Dette er den typiske masseprodusent.

Kostnadsledelse innebærer en optimal effektivitet i alle aktiviteter. Streng kontroll med kostnader er en kritisk suksessfaktor, samtidig som man må opprettholde et servicenivå som er godt nok i forhold til kundegruppen. Kjente eksempler på virksomheter med en kostnadslederstrategi finner vi først og fremst innen varehandelen, med bedrifter som IKEA og REMA 1000.

For en kostnadsleder er det viktig med informasjonssystemer som kan bidra til både kostnadskontroll og planlegging. Det betyr at en slik virksomhet trenger et godt økonomistyringssystem, og videre systemer for produksjonsplanlegging, innkjøp og distribusjon. Dette er systemer alle bedrifter trenger, men kostnadslederen trenger dem i større grad, og av bedre kvalitet, enn virksomheter med en annen strategi.

Kostnadsledere driver typisk med masseproduksjon, noe som innebærer høy grad av automatisering. Systemer for produksjonsplanlegging og produksjonsstyring er derfor sentrale. Også logistikken vil være av stor betydning, spesielt planlegging av effektiv distribusjon.

2.2.2 Differensiering

Selv om man fra media kan få inntrykk av at alle prøver å konkurrere på pris i dag, er det langt fra tilfelle. Med en differensieringsstrategi er det ikke lenger standardiserte produkter, men produkter som *skiller seg fra* konkurrentenes man tilbyr. Man kan også bruke service som et virkemiddel i stedet for produktet selv. Kunden er villig til å betale mer for å få et produkt eller en service som skiller seg fra konkurrentenes. Fortsatt er det imidlertid et massemarked man henvender seg til, selv om det ofte vil være smalere enn for de typiske lavkostprodukter. Kjøkkeninnredningsprodusenten Norema kan være et eksempel på en virksomhet med defferensieringsstrategi. Bilprodusenter som Mercedes og Audi brukes også ofte som eksempel (Roos 2002).

En virksomhet med differensieringsstrategi må selvsagt også ha systemer for økonomistyring og produksjonsplanlegging, men for disse virksomhetene vil også andre informasjonssystemer være av stor betydning. Spesielt systemer for salg og service kan være av stor betydning. Norema bruker for eksempel DAK-baserte salgssystemer hos forhandlerne. Disse systemene legger ordre dirkete inn i produksjonsplanleggingssystemet. Det totale systemet for produksjonsstyring og distribusjon sørger så for at et komplett kjøkken, inkludert hvitevarer, kan leveres innen to uker.

Vi kan vise to eksempler på samme type produkt, men med helt forskjellige kundegrupper (vi ser bort fra tidsdifferansen. Ford modell T til venstre nedenfor ble laget med «alminnelige amerikanere» som kundegruppe. Henry Ford ville at alle skulle ha råd til en T-Ford. Rolls Roycem nedenfor til høyre er rettet inn mot en helt annen kundegruppe – de mest velbeslåtte av oss.

Figur 2.2 Samme type produkt – to forskjellige forretningsstrategier (Ford modell T ca 1925 til venstre. Foto: Christoffer Burke, Wikomedia Commons https://commons.wikimedia.org/wiki/File:Ford_Model_T_1924-1925.jpg#file. *En Rolls Royce fra 1927 til høyre. Foto. Brett Weinstein, Wikipedia Commons* https://commons.wikimedia.org/wiki/File:1927_Rolls-Royce_Phantom_I.jpg)

Vi kan også ta med merkevarer som Levis som eksempel. Levis konkurrerer absolutt ikke på pris, men på at kunder er villige til å betale mer for dette merket (som jo også har en rekke forskjellige modeller). Tilsvarende varer som Henry Choise konkurrerer derimot på pris.

2.2.3 Fokusert strategi

En fokusert strategi innebærer at virksomheten konsentrerer seg om en bestemt kundegruppe. Denne kan være definert ut fra forskjellige kriterier: geografisk, inntektsmessig eller annet. Ved å konsentrere seg om en smal kundegruppe, kan man betjene denne bedre enn de virksomhetene som satser på et bredt marked. Ofte er også fortjenestemulighetene større i små markeder, fordi de store virksomhetene ofte ikke bryr seg om å gå inn i dem.

Enhver lokal virksomhet kan vi se på som fokusert; den betjener først og fremst det lokale markedet. Det er imidlertid virksomheter med et geografisk stort marked, som samtidig har en fokuseringsstrategi vi vanligvis omtaler som virksomheter med fokusert strategi. Som et typisk eksempel kan vi trekke frem bedrifter som Rolls Royce. Denne fabrikken satser på en relativt liten gruppe av svært rike mennesker. Concorde hadde også en fokusert strategi; det var ikke mange mennesker som var villige til (og hadde muligheten til) å betale opp til 100000 kr for en flytur over Atlanteren.

De fleste virksomheter med en fokuseringsstrategi vil samtidig satse på differensiering. I sin ekstreme form kalles en slik strategi en nisjestrategi: man fokuserer på et meget smalt marked, men et marked som samtidig er villig til å betale (typisk fordi det ikke er så mange alternativer i et smalt marked). Eksempler på smale markeder kan være markedet for clavicordstrenger og markedet for storformatkameraer.

Den fokuserte bedrift som satser i et globalt marked er et eksempel på en bedrift som har spesielt god nytte av internett. Mange av nettbutikkene er nettopp slike bedrifter.

Et eksempel på en virksomhet med et fokusert marked er nettbutikken til Really Right Stuff. Denne virksomheten er basert i California, men selger spesialisert fotoutstyr til kunder over hele verden. Deres spesialitet er hurtigkoblinger – disse brukes til raskt å montere kameraer på et stativ (og like raskt ta dem av igjen). Produktene er langt fra billige, men målgruppen betaler gjerne.

2.2.3 Blandingsstrategier

Selv om kostnadsledelse og differensiering i prinsippet er gjensidig utelukkende, er det i praksis noen bedrifter som til en viss grad har klart kombinasjonen. Et eksempel på dette er IKEA. Selv om IKEA er en kostnadsleder, er bedriften samtidig blitt en merkevare, og et "inneprodukt" hos store grupper.

Typisk i dag er at bruken av informasjonsteknologi gjør det mulig for mange bedrifter å kombinere differensiering med kostnadsledelse. IKEA bruker utvilsomt avanserte informasjonssystemer for å styre produksjon, innkjøp og distribusjon. Også Norema kombinerer differensiering med konkurransedyktige priser, selv om bedriften ikke kan konkurrere med for eksempel IKEA på pris.

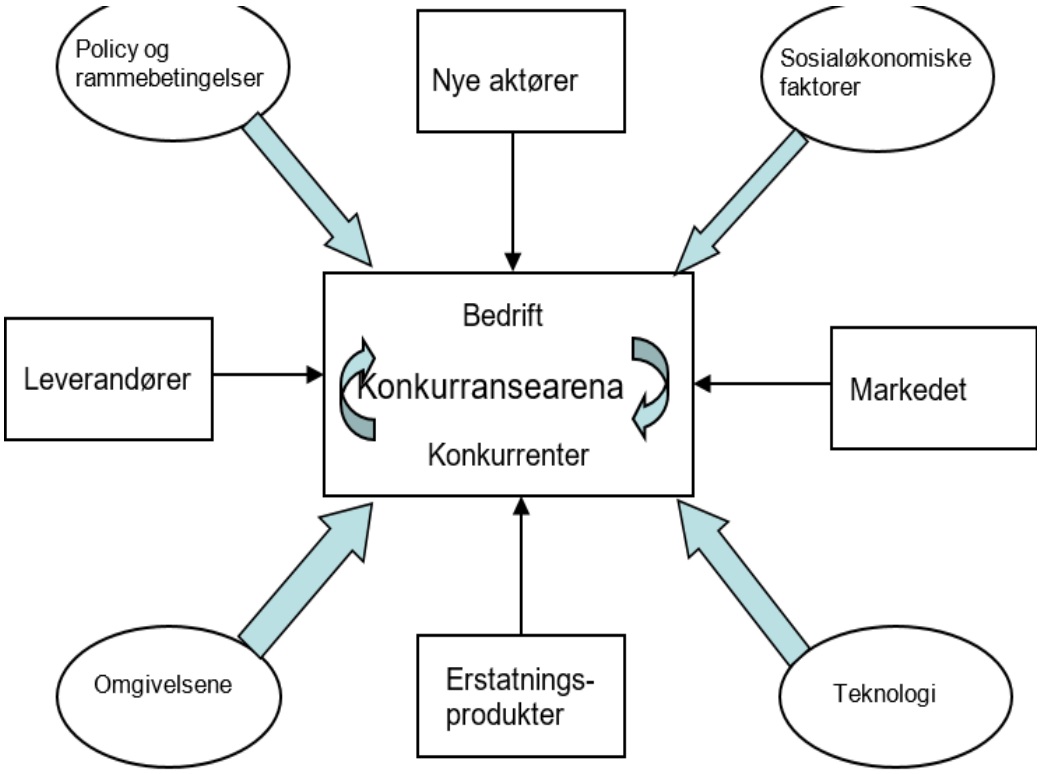
Det viktige med forretningsstrategier er at virksomheten må ha en klar forestilling om hvilken strategi den ønsker å legge opp til. Virksomheter som får problemer kjennetegnes ofte nettopp av at de mangler en klar strategi.

Vi skal i det videre se nærmere på modeller for strategisk analyse av en virksomhet. Det finnes mange slike modeller, og de har forskjellige perspektiver. De mest kjente er konkurransearenamodellen og verdikjedemodellen.

2.3 Konkurransearenaanalyse

Harvard-professoren Michael Porter brukes ofte i forbindelse med strategiske analyser, spesielt knyttet til strategisk bruk av informasjonsteknologi. Det er to analysemodeller som er knyttet til Porter: konkurransearenamodellen og verdikjedemodellen. Spesielt den siste er sterkt knyttet til arbeidet med å utvikle strategier basert på informasjonsteknologi.

Konkurransearenamodellen (Porter 1980) er en modell for å analysere bedriftens ytre betingelser:



Figur 2.3 Konkurransearenamodellen (Etter Haraldsen 1998)

Konkurransearenamodellen fokuserer på fem faktorer (Porter 1980, Haraldsen 1998, Roos 2002):

- Konkurransearenaen, som består av bedriften og dens konkurrenter
- Markedet, kjennetegnet ved kundenes forhandlingsposisjon
- Leverandører, kjennetegnet ved deres forhandlingsposisjon
- Nye aktører, eller inntrengere
- Erstatningsprodukter, eller substitutter.

Konkurransearenaen er scenen for konkurransen mellom etablerte bedrifter i samme bransje. Hvordan konkurransen arter seg er avhengig av en rekke faktorer, som markedets størrelse, antall aktører, bransjens vekst og kostnadsnivå i bransjen. Innen konkurransearenaen vil vi finne eksempler på alle forretningsstrategiene vi har gått gjennom. I tillegg er det som vi ser andre faktorer som bestemmer konkurransen.

I bransjer med høye investeringer i produksjonsutstyr, slik som oljebransjen og kjemisk industri, vil man så sant det er mulig utnytte produksjonskapasiteten fullt ut. Den samlede produksjonen vil derfor bli stor (en del av forklaringen på at OPEC som regel produserer mye mer olje enn kartellet har bestemt at de skal gjøre).

Er antall aktører høyt, vil vi nærme oss en tilstand sosialøkonomene kaller fullkommen konkurranse. Ser vi bort fra politiske reguleringer, vil denne konkurranseformen føre til de laveste prisene. Er det derimot få aktører, nærmer vi oss et oligopol. Fra sosialøkonomien vet vi at konkurransen da ikke er så effektiv, og prisene blir høyere. Med et monopol er det

helt mangel på konkurranse. Som kjent var NRK er monopol før satellittenes tid (selv om mange kunne ta inn svensk TV).

Erstatningsprodukter, eller substitutter, betyr i denne sammenheng nye produkter som kan erstatte eksisterende. Et eksempel er da elektroniske kalkulatorer erstattet mekaniske regnemaskiner, eller da elektroniske klokker erstattet de mekaniske. I dag har digitale kameraer helt overtatt for filmbaserte, mens smarttelefoner er i ferd med å utkonkurrere kompakte digitalkameraer. Dvd-opptagere overtok markedet for videoopptagere, for nå å være presset ut av Blue-Ray. For en bedrift er det viktig å overvåke markedet for å identifisere potensielle substitutter, slik at tiltak kan treffes før det er for sent. Den økonomiske historien er full av eksempler på bedrifter og hele bransjer som ikke har klart dette. Den sveitsiske klokkeindustrien holdt for eksempel på å gå under da de japanske elektroniske klokkene slo igjennom.

Markedet, som består av kundene, og leverandørene, er begge grupper karakterisert ved sin forhandlingsmakt. Med dette menes i hvilken grad disse gruppene er i stand til å forhandle med bedriften om priser og andre betingelser. Mens kundene er interesserte i å presse prisene ned, vil leverandørene ha den motsatte interessen. Hvordan en bedrift klarer seg i denne situasjonen er avhengig av en rekke faktorer. For bedriften er det særlig viktig å analysere situasjonen. Dette skjer typisk ved at man analyserer kundenes kjøp. Man er ute etter både hvem som kjøper hvilke produkter, når de kjøpes, hvilke produkter man tjener mest og minst på, hvilke produkter som er i vekst og hvilke som er i nedgang osv. Ved å analysere kundenes valg av de enkelte leverandørenes produkter, kan man kanskje finne frem argumenter til bruk i forhandlinger med disse.

Bedriftens overvåkning av konkurransesituasjonen krever analyse av data fra en rekke forskjellige kilder. Dette er nettopp en av de viktigste bruksområdene til de informasjonssystemene vi skal konsentrere oss om i det resterende.

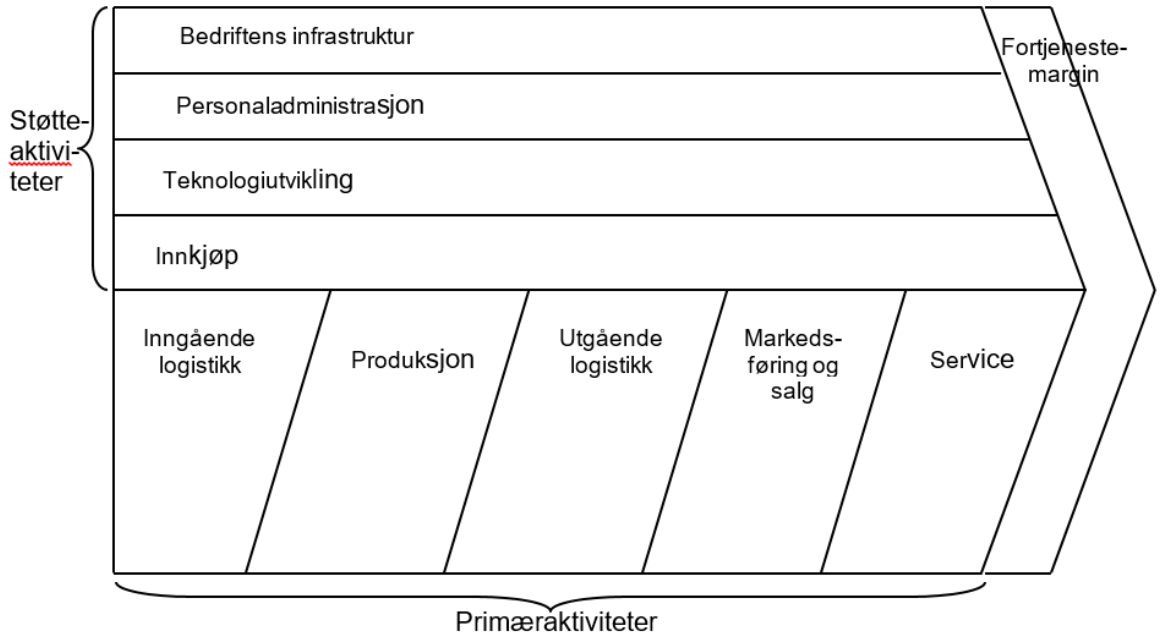
2.4 Verdikjedeanalyse

Verdikjedemodellen (Porter 1985) er laget for å analysere indre betingelser i organisasjonen. Utgangspunktet for verdikjedemodellen er verdiskapingen, det vil si de prosessene i organisasjonen som skaper verdier. Porter sier at verdiene blir skapt gjennom identifiserbare verdiskapende aktiviteter. Disse aktivitetene henger sammen i det han kaller organisasjonens verdikjede. Alle aktivitetene i kjeden bidrar til den endelige verdiskapingen.

I organisasjonens verdikjede skiller Porter mellom primæraktivitetene, som er de som egentlig skaper verdier, og støtteaktivitetene. De siste bidrar ikke direkte til verdiskapingen, men er nødvendige for at primæraktivitetene skal kunne utføres.

Når en organisasjon fremstiller en vare eller en tjeneste og bringer denne frem til sluttbruker, er det flere identifiserbare aktiviteter som inngår i denne prosessen. Det er disse aktivitetene som er primæraktivitetene, og altså utgjør organisasjonens verdikjede. Porter identifiserte fem forskjellige primæraktiviteter i det han kalte en generell verdikjede:

- logistikk inn
- produksjon eller behandling
- logistikk ut
- markedsføring
- salg og service

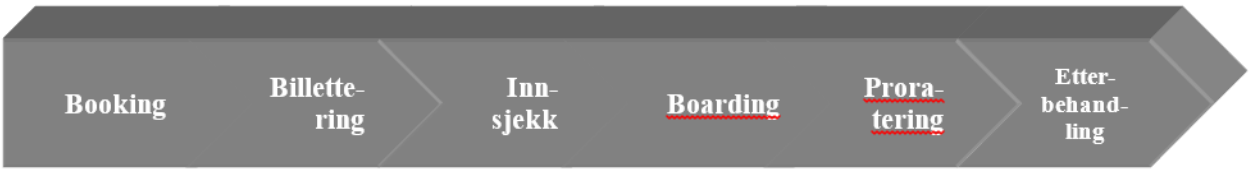


Figur 2.4 Verdikjedemodellen (Etter Haraldsen 1998)

Alle disse aktivitetene bidrar på sin måte direkte til organisasjonens verdiskaping. De aktivitetene Porter bruker i sin generelle verdikjede, er imidlertid bare et utgangspunkt for det analysearbeidet som er nødvendig i en strategiprosess. Den enkelte organisasjon må gå gjennom sine egne aktiviteter for å identifisere:

- Hvilke verdiskapende aktiviteter organisasjonen faktisk har (og dette kan avvike fra den generelle verdikjeden)
- Hvor mye den enkelte aktivitet bidrar til den totale verdiskapingen, det vil si hvor god organisasjonen er til å utføre hver enkelt aktivitet

Verdikjedene kan se svært forskjellige ut for forskjellige organisasjoner. Det er imidlertid slik at inne samme bransje vil verdikjedene være nokså like fra organisasjon til organisasjon. Dette forklares av institusjonell teori, som vi har sett på tidligere. For eksempel for passasjertrafikk med et flyselskap vil verdikjeden kunne se slik ut:

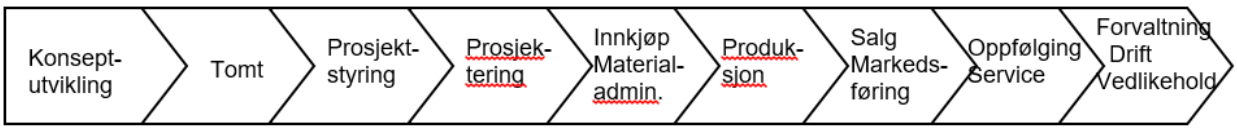


Figur 2.5 Verdikjede for et flyselskap (etter Heie 2003)

Vi kjenner her igjen de fleste begrepene fra våre egne erfaringer som flypassasjerer:

- Booking er bestillingen av billettene, som kan skje gjennom tre forskjellige sentraler:
    - Telefonbestilling fra selskapets call-senter
    - Gjennom reisebyrå eller andre byråer (Norwegian tilbyr billettkjøp hos Narvesen)
    - Over web, noe som blir mer og mer vanlig
- Billettering er utskrift og annen behandling av billetten (som i økende grad er elektronisk)
- Innsjekking er det som skjer når passasjeren ankommer flyplassen. Har man bagasje skal man først sjekke inn den. Bagasjen skal så håndteres videre.
- Boarding er når passasjerer registreres ved gate'n når man skal gå om bord i flyet.
- Proratering er fordeling av inntekter mellom forskjellige aktører involvert i en reise. Mange reiser krever skifte av fly for forskjellige strekninger. Innenlands er det mange reiser som involverer for eksempel SAS/Brathens og Widerøe. I Norden opererer SAS som egne selskaper i Sverige og Danmark, og har også et eget selskap for internasjonale flyvninger. Man samarbeider også med andre selskaper som KLM og British Airways om internasjonale ruter. Passasjeren betaler imidlertid én billett for hele reisen.
- Etterbehandling omfatter en rekke aktiviteter som klagebehandling.

Vi skal ta med et eksempel til. Figur 2.9 viser verdikjeden for en entreprenørbedrift (Reve 1989). Bedriften retter seg mot boligbygg og yrkesbygg, og har 9 trinn i primærkjeden.



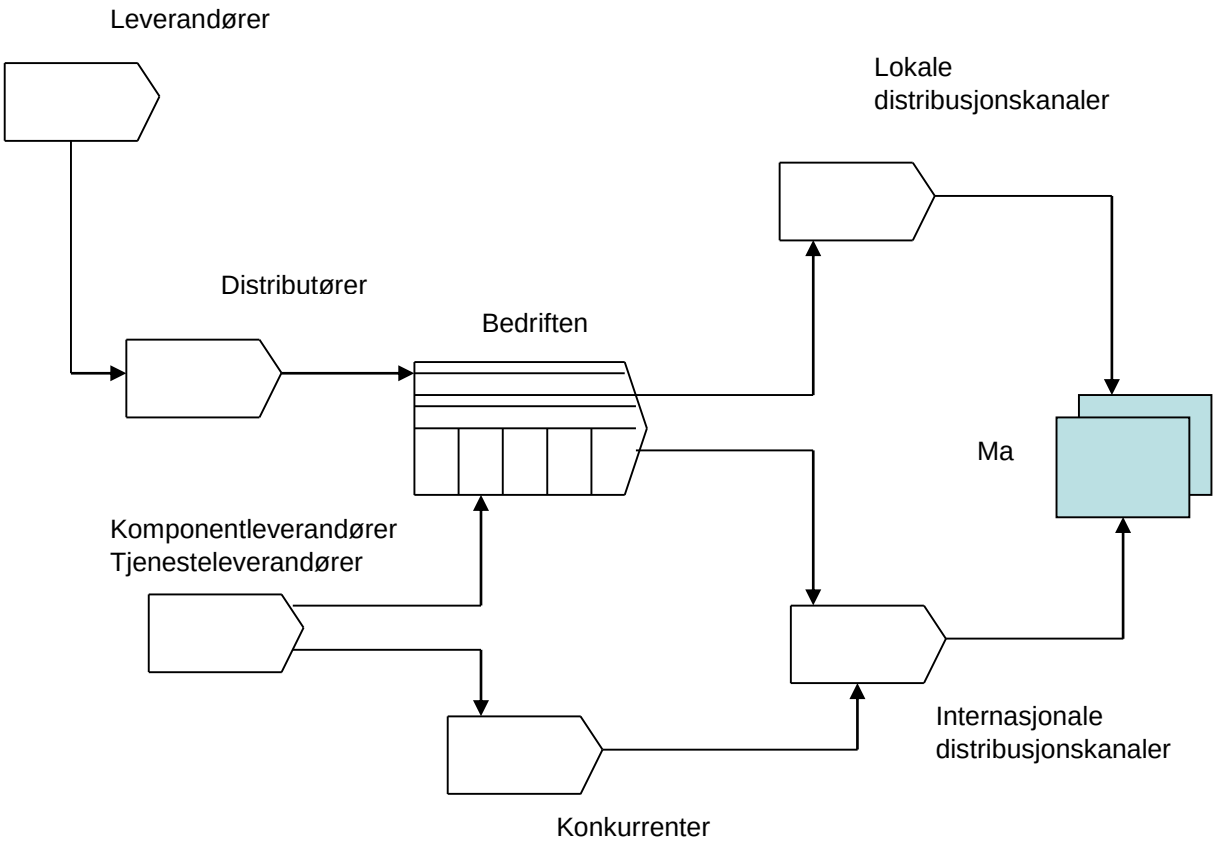
Figur 2.6 Verdikjede for entreprenør (etter Reve 1989)

Hensikten med analysen av bedriftens interne verdikjede er altså på den ene siden å kartlegge hva man egentlig gjør, og på den annen side å fastslå «hvor flinke vi er». Målet er å finne frem til bedriftens særegne kompetanse. Den særegne kompetansen er selve grunnlaget for bedriftens eksistens. Med særegen kompetanse siktes til det bedriften er flinkere til enn sine konkurrenter. Meningen er at en analyse av verdiskapingen i de enkelte verdiskapingsaktiviteter i bedriften skal avsløre dette. Når man har funnet frem til hvor bedriftens særegne kompetanse ligger, står så bedriften overfor flere strategiske valg:

- Hvordan skal vi ta vare på og videreutvikle denne kompetansen?
- Hvordan beskytter vi vår særegne kompetanse?
- Hva gjør vi med de aktiviteter der vi ikke er flinke?

Det siste punktet må ses i lys av verdisystemet eller den eksterne verdikjeden. Denne består av interne verdikjeder i alle de bedriftene som er involvert fra produksjon frem til kunden.

Den enkelte bedrift må ta stilling til hvilke aktiviteter den vil utføre selv, hvilke den vil utføre i samarbeid med andre og hvilke den vil kjøpe. Videre er det et strategisk valg for bedriften hvordan den vil oppnå kontroll over produktets verdikjede. Kontroll med produktets verdikjede er det samme som å ha kontroll med egen situasjon.



Figur 2.7 Verdisystemet eller den eksterne verdikjeden (etter Haraldsen 1998)

I dag er integrasjon i verdisystemet blitt vanlig i mange bransjer. Slik integrasjon kan skje på forskjellige måter. En måte er oppkjøp eller fusjonering. Innen for eksempel dagligvarebransjen har flere av de store kjedene kjøpt opp distribusjonsleddet (grossisten). En annen måte er å inngå omfattende avtaler med andre aktører, der bedriftene forplikter seg ut over vanlig kjøp og salg. Elektronisk handel basert på EDI er et eksempel på dette (mer om dette i et senere kapittel). Typisk er at bedriftenes informasjonssystemer integreres for å effektivisere samhandlingen mellom aktørene.

Også når det gjelder støtteaktivitetene kan den samme analysen gjøres. Porter identifiserer her fire kategorier støtteaktiviteter:

- innkjøp
- teknologi og produktutvikling
- personale
- organisasjon og ledelse

Igjen må den enkelte organisasjon identifisere hvilke støtteaktiviteter den selv har, og hvor effektivt de utføres. Det sentrale med støtteaktivitetene er at de ikke bidrar direkte til den pris man kan få i markedet, og effektivitetskravene er derfor strenge.

Det finnes en del hovedregler for hvordan bedriften bør opptre i forhold til grad av kontroll over verdikjeden. Den viktigste regelen er at aktiviteter som bygger på bedriftens særegne kompetanse, skal bedriften utføre selv. Dette er kompetanse som må beskyttes mot konkurrenter.

Når det gjelder andre aktiviteter vil en god regel være at aktiviteter basert på standardteknologi og kompetanse, kan bedriften la andre utføre. Det vil alltid være andre som kan utføre disse aktivitetene billigere enn bedriften selv. Andre aktiviteter vil stå i en mellomklasse idet de er basert på teknologi og kompetanse som ikke er standard, men som bedriften selv ikke behersker i tilstrekkelig grad. Dette er aktiviteter som bedriften kan velge å utføre i samarbeid med andre.

Verdikjedeanalyse trekkes spesielt frem i forbindelse med bruk av informasjonsteknologi. Årsaken til dette er at det er utveksling av informasjon mellom leddene i verdikjeden, og mellom primærkjeden og støtteaktivitetene. Bruk av informasjonsteknologi kan gjøre det mulig å velge helt nye strategier. Intern effektivitet vil i stor grad være basert på utnyttelse av kompetanse og informasjonssystemer. Verdikjedeanalysen vil derfor være et verktøy for å finne frem til strategier basert på informasjonsteknologi og til å finne frem til hvordan informasjonssystemer kan styrke allerede valgt strategi.

2.5 Å lage en IT-strategi

Som vi har vist vil virksomhetens IT-strategi bygge på analyser av selve forretningsdriften. Informasjonsteknologien skal støtte eller muliggjøre den daglige drift, og gjøre det mulig å analysere egen virksomhet fortløpende. Dette krever investeringer i maskinvare, programvare og opplæring.

I en ideell verden ville det vært slik av det nå bare var å sende ut bestillinger. Virkeligheten er imidlertid annerledes. I den virkelige verden vil økonomien spille en stor rolle – de fleste virksomheter har rett og slett ikke råd til å gjøre alle disse investeringene samtidig. Selv om det hadde vært mulig, er det også andre begrensninger. Innføring av informasjonsteknologi i en virksomhet er noe ganske annet enn å installere et program på egen PC. Vi snakker om prosjekter som kan ta fra måneder til år å gjennomføre. I noen tilfelle skyldes det at programvaren må utvikles spesielt for virksomheten. Uansett vil alle prosjekter inneholde et vesentlig element av organisasjonsendring. Med innkjøpt programvare sier vi ofte at prosjektet er 20 % teknologi og 80 % organisasjonsendring. Som om ikke dette var nok, må man heller ikke glemme opplæring av brukerne. At denne blir gjennomført skikkelig kan bety forskjellen mellom suksess og fiasko. Og alt dette skal gjennomføres mens den daglige drift går uavbrutt!

Som vi skjønner, er det nødvendig med omfattende planlegging av IT-prosjekter. Og det er nødvendig å ha en overordnet IT-strategi, som sier hvilke systemer som skal brukes av vår virksomhet for å nå målene. Ut fra IT-strategien lager man en IT-plan, som er en prioritert liste over systemer og når de skal være på plass.

En IT-strategi består typisk av tre delstrategier (Christensen 1999):

- Applikasjonsstrategi
- Teknologisk strategi
- Styringsstrategi

Applikasjonsstrategien er den sentrale delstrategien. Det er denne som sier hvilke systemer virksomheten skal ha. Analysemetodene som er gjennomgått tidligere i dette kapittelet er blant hjelpemidlene som brukes for å komme frem til dette. Det er imidlertid ikke bare et spørsmål om hvilke systemer man skal ha, men også om hvordan man skal få dem.

For det første vil virksomheten sannsynligvis allerede ha systemer i bruk. Det første spørsmålet man må stille er derfor om man kan bygge videre på disse, eller om man må ha noe helt nytt. Deretter kommer spørsmålet om man kan kjøpe det man trenger ferdig, eller om det er nødvendig å få laget noe spesielt til denne virksomheten. Når vi her snakker om å kjøpe ferdige systemer, er det ikke en enkel affære, noe artikkelen nedenfor fra Computerworld Norge skulle vise (SAP er et ferdig system som er laget for å kunne tilpasses den enkelte virksomhet).

Spørsmålet om man kan bygge videre på det man har avhenger delvis av hvor bra det man allerede har er, men også på om det er basert på fremtidsrettet teknologi. Det bringer oss over til neste delstrategi.

Teknologistrategien går ut på hvilken teknologisk plattform systemene skal baseres på. En virksomhets systemer kan baseres på et enkelt nettverk med PC'er eller Mac-maskiner i et nettverk, der en tjenermaskin brukes til å lagre data. Dette er slik det vanligvis gjøres i studentenes nettverk på en høyskole. I en virksomhet av noen størrelse er som regel ikke dette nok. Teknologien må for eksempel tillate mange samtidige brukere, og disse skal kanskje kunne bruke systemet fra hvor som helst utenfor selve virksomheten. Vi snakker da om maskintyper og ikke minst operativsystemer som kan håndtere dette. Videre kommer telekommunikasjoner inn for fullt.

Styringsstrategien for virksomhetens informasjonssystemer går på hvordan både drift og utvikling skal ledes. Her er det mange muligheter for organisering. Man kan for eksempel ha en egen IT-avdeling som driver med både drift og utvikling. Dette er den klassiske modellen. I dag er det imidlertid vanlig at den daglige driften av systemene er satt ut til en datasentral. Noen virksomheter utvikler sine egne systemløsninger, andre kjøper dem. I dag er det flere virksomheter som bruker helt nettbaserte systemer. Vi snakker da om at de ligger i "skyen" ("Cloud computing").

En viktig del av styringsstrategien er også kravet til kompetanse og opplæring. Hvor mye kompetanse på selve systemene og teknologien skal vi ha selv? Mange virksomheter har her valgt å sette ut drift og utvikling, men har selv bygget opp kompetanse på nettopp IT-strategi. Av stor betydning er også brukernes kompetanse på bruk av systemene. Mange feil skyldes manglende opplæring av brukerne. Her er det igjen et spørsmål om hvordan brukeropplæring skal skje: egen kursavdeling, kjøp av kurs eller intern opplæring ved bruk av såkalte superbrukere.

IT-strategien skal resultere i konkrete handlingsplaner. Det vanlige er at disse har en varighet på 3 til 5 år.

Kapittel 3 Beslutninger

En sentral oppgave for ledere er å ta beslutninger. Mange vil regne beslutningsdyktighet som den viktigste enkeltegenskapen for en leder. I moderne organisasjoner er myndighet til å ta beslutninger delegert nedover i organisasjonen, med mer eller mindre klare grenser for den enkelte leders myndighetsområde. Det er også vanlig at beslutninger er noe man arbeider seg frem til i grupper, enten ved konsensus (enighet) eller flertallsavstemninger.

Vi skal i dette kapittelet se nærmere på hvordan beslutninger tas, og hvilke forutsetninger som kreves for å ta dem. I denne sammenheng er riktig informasjon en kritisk faktor.

3.1 Rasjonalitet i beslutninger

Et grunnleggende spørsmål angående beslutninger er om de er rasjonelle, det vil si basert på fornuft, eller ikke. Man skulle umiddelbart tro at målet for en beslutningstager er å ta en beslutning basert på bruk av fornuft, men så enkelt er det ikke. For det første må vi spørre hvem beslutningen er rasjonell for: beslutningstageren eller organisasjonen. For det andre kan det godt hende at beslutningstageren er styrt av underbevisste holdninger. For det tredje krever rasjonalitet at man har tilgang på all relevant informasjon og er i stand til å behandle denne på en nøytral måte.

Rasjonalitet var forutsatt i de klassiske organisasjonsteoriene. Beslutningstagerne ble betraktet som rasjonelle vesener som tok beslutninger ut fra hva som var økonomisk lønnsomt. Dette perspektivet på beslutninger kalles derfor gjerne ”Economic man” – det økonomiske menneske (Busch 1995). Beslutningsprosessen ville være som følger:

- problemet er definert på forhånd
- det er et gitt sett alternativer
- til hvert alternativ er det et gitt sett konsekvenser
- alternativene kan rangeres ut fra bidrag til å nå organisasjonens mål, og det som da kommer best ut blir valgt
- man er ute etter den beste løsningen. Dette kalles optimering

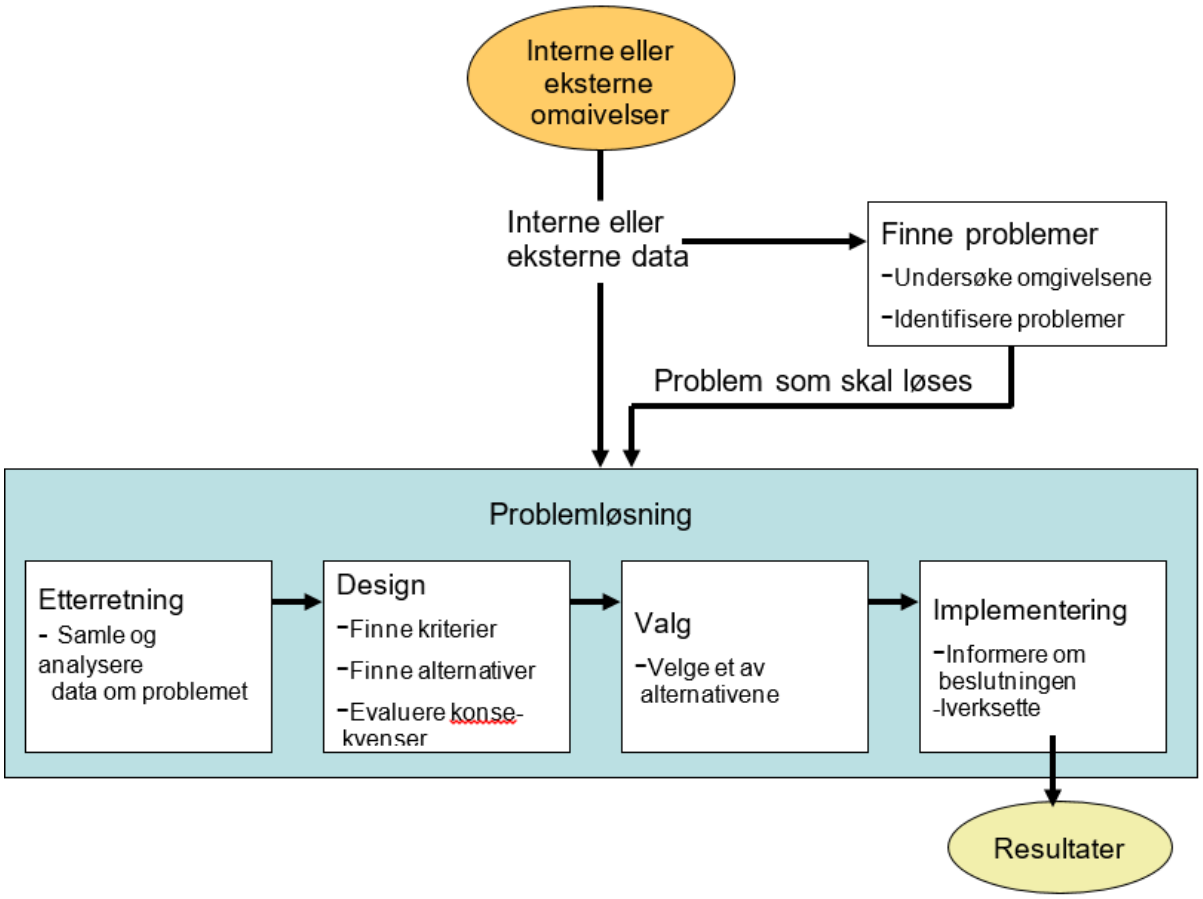
Dette er hva vi kaller en normativ modell, det vil si at den beskriver hvordan en beslutningsprosess burde være. Hvordan de er i virkeligheten er en annen sak. Det kan selvsagt også diskuteres om denne modellen er en god norm.

Det er mange svakheter med den rasjonelle modellen. For det første er det å definere problemet ofte svært vanskelig. Videre er det umulig å kjenne alle alternativer, og enda mer umulig å beregne alle konsekvenser. Dermed blir det i praksis ikke mulig å rangere alternativer ut fra bidrag til å nå målene, som det for øvrig også kan være uenighet om.

Herbert Simon utarbeidet en modifisert modell for beslutningsprosesser, kalt begrenset rasjonalitet. Etter denne modellen skal man tilstrebe rasjonalitet, samtidig som man erkjenner at full rasjonalitet ikke er mulig. Forutsetningene for Simons modell, som han kalte ”Administrative Man”, er disse:

- organisasjonen må søke etter og definere problemer
- man kan aldri finne alle alternativer, men vil i stedet søke til man finner et som er bra nok
- man kan ikke beregne alle konsekvenser, men må finne de man kan håndtere i organisasjonen
- man må bruke enkle tommelfingerregler for å vurdere alternativene mot hverandre, fordi organisasjonens mål er så sammensatt at det er umulig å ta hensyn til alle
- man er ikke ute etter den beste løsningen, men en som er god nok. Dette kalles satisfiering

Simons modell for begrenset rasjonalitet i beslutninger har fått stor betydning. Ut fra denne kan man sette opp en modell for hele beslutningsprosessen (Alter 2002):



Figur 3.1. Trinnene i beslutningsprosessen (etter Alter 2002)

Modellen viser beslutningstagning som en prosess med å identifisere problemer etterfulgt av en problemløsningsprosess.

Finne problemer: identifisere og formulere problemer som må løses. Dette er selve grunnlaget for en god beslutning, fordi det alltid vil være en risiko for at man forsøker "å løse galt problem". Identifikasjon av problemer vil være på grunnlag av formulerte mål for organisasjonen og informasjon om den virkelige tilstanden

Problemløsningen består så av fire trinn:

- **Etterretning (Intelligence):** Innsamling og analyse av relevante data om problemet. Disse data er grunnlaget for neste fase
- **Utforming (Design):** Formulering av alternative løsninger og analyse av konsekvenser
- **Valg:** I denne fasen skal det beste alternativet plukkes ut. Ofte vil man her analysere mer i dybden et par av alternativene fra utformingsfasen. Valget vil som regel inkludere mange personer i organisasjonen, gjerne i form av grupper som skal komme frem til valget
- **Implementering:** Nå skal beslutningen settes ut i livet. Beslutningen må kommuniseres til organisasjonens medlemmer og interessenter, og man starter gjennomføringen av forskjellige tiltak. Det er viktig at beslutningen kan begrunnes slik at det skapes forståelse for den også blant de som ikke var direkte involvert i prosessen, men som er viktige for gjennomføringen

I alle fasene er informasjon sentralt, og forskjellige typer informasjonssystemer kan brukes for å støtte prosessen. Det er derfor viktig å forstå hvordan beslutninger tas for å forstå hvordan informasjonssystemer kan brukes av beslutningstagere.

Modellen viser beslutningsprosessen som strengt linjær, der en fase følger etter at en annen er avsluttet. I virkeligheten vil det være iterasjoner hele veien, det vil si at man ofte må gå tilbake for å gjøre deler av jobben om igjen.

3.2 Andre typer beslutninger

Både Economic Man-modellen og Administrative Man bygger på rasjonalitet. Vi kan si at disse modellene er normative, det vil si at de beskriver hvordan beslutningsprosesser bør være. De er også analytiske. Det finnes imidlertid også andre beslutningsmodeller som ikke bygger på at beslutninger tas ut fra rasjonalitet. Disse modellene kan vi si er mer deskriptive, det vil si at de forsøker å forklare adferd vi faktisk ser. De er ikke analytiske. Vi skal kort nevne slike modeller. Det er viktig å forstå at ingen av disse modellene gjør krav på å forklare hele virkeligheten. Hver av dem kan brukes til å forklare deler av det vi ser i organisasjoner.

3.2.1 Politiske beslutningsmodeller

I politiske beslutningsmodeller eksisterer ikke noen felles mål i organisasjonen. Forskjellige aktører har forskjellige mål, som ofte kommer i konflikt. Dette er en situasjon vi kjenner fra politikken, der forskjellige grupper i samfunnet vil ha uforenelige mål. Den samme situasjonen finner vi også i organisasjoner. Det finnes der ikke politiske partier som kan bære frem forskjellige meninger, men det finnes både formelle og uformelle strukturer for det samme. Fagforeninger et et eksempel på formelle organer som er representert i den interne politiske tautrekkingen i en organisasjon. Politiske prosesser vil typisk komme til uttrykk i avstemninger.

3.2.2 Søppelbøttemodellen

Søppelbøttemodellen – kanskje bedre kjent som Garbage Can-modellen – beskriver beslutninger som et møte mellom et problem og en akseptabel løsning. Søppelbøtten er selve beslutningssituasjonen, som deltagerne i prosessen slipper både problemer og løsninger oppi.

Også denne modellen kan beskrive virkelige beslutningsprosesser på en god måte. Det er velkjent i mange organisasjoner at problemer kan flyte rundt i lang tid uten at noen tar tak i dem. Det er heller ikke uvanlig at noen har ideer det ikke blir grepet tak i, fordi det ikke er oppfattet noe problem som ideene kan løse.

3.2.3 Inkrementalistmodellen

I følge inkrementalistmodellen vil ledere redusere usikkerhet ved å velge alternativer som er bare litt forskjellig fra tidligere løsninger. Lederne vil sjelden ta beslutninger som skiller seg radikalt fra det de har gjort tidligere. En slik beslutningsmodell har fått et ironisk navn: ”The Science of Muddling Through” (”vitenskapen om å rote seg frem”). Essensen i inkrementalistmodellen er at beslutningstageren reduserer risiko ved å gå forsiktig frem. Dette fungerer imidlertid best når omgivelsene er nogenlunde stabile (Jones 2004).

3.2.4 Den ustrukturerte modellen

Når det er hyppige og brå forandringer i omgivelsene, er inkrementelle beslutninger ikke spesielt velegnet. Det å gå forsiktig frem kan i slike tilfelle være et hinder for nødvendig tilpasning. Den ustrukturerte modellen skal beskrive hvordan beslutninger kan tas under høy usikkerhet (Jones 2004). Under normale forhold vil beslutninger tas inkrementelt, det vil si at en rekke små beslutninger bygges opp til en stor. Når organisasjonen plutselig blir stilt overfor et stort og uventet problem, skal man imidlertid bryte med den inkrementelle modellen. Man går da ”tilbake til tegnebrettet”, og starter forfra med beslutningen (ibid.). Usikkerhet I omgivelsene tvinger beslutningstagere til stadig å finne nye løsninger på problemer, men disse løsningene kan ofte bli tatt på en tilfeldig måte.

3.3 Klassifikasjon av beslutninger

Beslutningene i en organisasjon er av forskjellig slag. Det er åpenbart at det er forskjell på en beslutning om man skal sette to eller tre personer til å betjene kassene i en butikk tirsdag formiddag, og en beslutning i samme bedrift om man skal etablere nye butikker i Øst-Europa. Det er derfor laget modeller for klassifisering av beslutninger.

3.3.1 Beslutningsnivå

En vanlig måte å klassifisere beslutninger på er etter hvor i organisasjonen de tas. Organisasjonen ses da som et hierarki i form av en pyramide.

Det nederste laget i pyramiden er det som produserer organisasjonens varer og tjenester, og støtteoppgaver direkte tilknyttet dette. Dette nivået kalles det **operative** nivå. Beslutningene her skal først og fremst sørge for at tilgjengelige ressurser blir utnyttet.

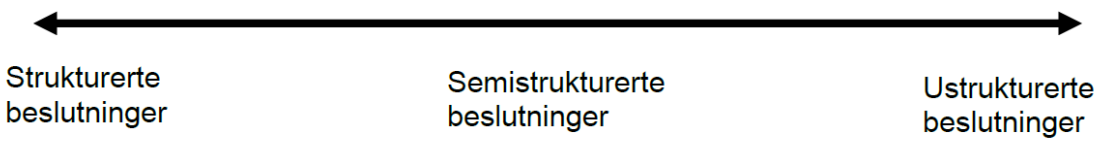
Over dette nivået kommer et nivå der oppgavene er knyttet til å skaffe ressurser, og kontrollere at disse blir utnyttet optimalt. Dette nivået kalles gjerne det **administrative** eller det **taktiske** nivået. Siden beslutningstagerne på dette nivået er det vi gjerne kaller mellomledere (managers på engelsk) kan vi også kalle det **mellomledernivået**.

Toppen av pyramiden utgjøres av toppledelsen (executives på engelsk). Deres oppgave er først og fremst å peke ut retningen organisasjonen skal gå i. Sentrale spørsmål for disse er hvilke produkter/tjeneste man skal tilby i hvilke markeder, og hvilke samarbeidspartnere man skal ha. Slikt arbeid kalles strategisk, og dette nivået kalles det **strategiske** nivået.

Figur 3.2 Klassisk organisasjonspyramide (etter Laudon 2017)

3.3.2 Beslutningstype

Noen beslutninger kan man ta ved å følge klare regler. Et eksempel på dette kan være for innkjøp av varer i butikk: når det er tomt i hylla, bestiller vi en kartong med varen (dette er selvsagt en altfor enkel regel, men den illustrerer poenget). Ved andre beslutninger er det ikke slike regler som kan følges. Da jernteppet falt rundt 1990 var det ingen i Øst-Europa som visste hvordan de skulle gå frem for å skape demokratiske samfunn med en markedsøkonomi. I andre tilfelle kan man bruke kjente regler på deler av beslutningen, mens det ikke finnes regler for andre deler. En beslutning om å etablere norske butikker i Øst-Europa (som mange av butikk-kjedene har gjort) kan være et eksempel. Deler av en slik beslutning kan bygge på kjente regler for å beregne forventet avkastning på kapital, men andre deler vil være skritt ut i det ukjente. Vi kaller disse forskjellige typene beslutninger strukturerte, ustrukturerte og semistrukturerte. Vi kan se disse typene som plassert på en skala, som vist i figur 3.3.



Figur 3.3 Skala for beslutningstyper (etter Sharda 2014)

En virkelig beslutningssituasjon kan ligge hvor som helst på skalaen, selv om fullstendig ustrukturerte beslutningssituasjoner er svært sjeldne.

Strukturerte beslutninger er altså beslutninger der man kan følge klare regler. I en fullstendig strukturert beslutningssituasjon (helt til venstre på skalaen i figur 6.3) kan man derfor godt la et datamaskinprogram ta beslutningen. Et typisk system for lagerstyring vil for eksempel lage forslag til innkjøp. I de siste årene er det også kommet systemløsninger der programmet faktisk også tar avgjørelsen.

I en organisasjon er strukturerte beslutninger typisk på det operative nivået, selv om det også vil forekomme semistrukturerte situasjoner der. Det normale er imidlertid at situasjoner det ikke finnes regler for skal sendes oppover til neste nivå (jfr Scientific Management).

Ustrukturerte beslutninger er altså beslutninger der det ikke finnes noen regler eller tidligere erfaringer man kan bruke. En ren ustrukturert beslutningssituasjon er uhyre sjelden, det finnes som regel noen erfaringer man kan legge til grunn. Ustrukturerte beslutninger vil typisk være knyttet til dramatiske og uventede hendelser. Jernteppets fall er allerede nevnt som eksempel. Andre eksempler kan være situasjonen mange firmaer stod i etter at World Trade Center raste sammen. Et tredje eksempel er Apollo 13 – situasjonen, skildret i en film med Tom Hanks i hovedrollen. At man klarte å bringe romskipet tilbake til Jorden med hele mannskapet i live var en bragd av de helt store.

Ikke minst har vi ferske erfaringer med pandemien forårsaket av Corona-viruset. Mer om det nedenfor.

Ustrukturerte beslutninger, i den grad de forekommer, vil være toppledelsens ansvar. En slik situasjon krever mye når det gjelder utforming av alternativer, og ikke minst i oppfølgingen av iverksettingen.

Semistrukturerte beslutninger omfatter svært mange av de beslutninger som tas i en organisasjon. De fleste beslutningssituasjoner vil befinne seg et sted på skalaen mellom ytterpunktene, selv om mange av disse nok vil ligge nær den strukturerte enden.

I en semistrukturert situasjon vil man delvis bruke kjente regler, som investeringsmodeller, og delvis kreativitet i å finne løsninger. De fleste beslutninger på mellomledernivå vil være semistrukturerte, og da fra omtrent midten av skalaen og mot den strukturerte enden. Også på strategisk nivå vil de beslutningene stort sett være semistrukturerte, men nå fra omtrent midten og mot den ustrukturerte enden.

Business Intelligence handler om å bruke informasjonsteknologi i beslutningsprosesser.

Et tidlig rammeverk for forståelse av bruk av IT i beslutninger er Gorry og Scott-Mortons fra 1971, som er gjengitt i læreboken. Denne settes opp som en matrise med organisasjonsnivå som en akse og grad av struktur som den andre:

	Type kontroll		
Type beslutning	Operasjonell	Mellomleder	Strategisk
Strukturert	Ordreregistrering Fakturering Betaling	Budsjettanalyse Personalrapporter Kortsiktige prognoser	Finansstyring Investering
Semistrukturert	Produksjonsplanlegging Lagerstyring	Kredittevaluering Budsjettering Produksjonsplanlegging Kategorisere inventar	Bygge nytt anlegg Fusjoner og oppkjøp Kvalitetssikring Personalpolitikk
Ustrukturert	Kjøp av programvare Godkjenne lån Operere hjelpetjeneste	Forhandlinger Rekruttere lederstilling Kjøpe hardware Lobbying	Planlegge forskning og utvikling (R & D) Utvikle ny teknologi

Figur 3.4 Grunnlag for beslutninger (etter Sharda 2014)

Coronaviruspandemien

Vinteren 2020 stod I Corona-virusets tegn. Det som startet som en epidemi i Wuhan i Kina, spredte seg og ble snart erklært som en pandemi av WHO. Det store problemet var at dette var et nytt virus, som det ikke fantes vaksine mot. Dødeligheten for smittede var også relativt høy, spesielt blant de over 70.

Det var opp til det enkelte lands myndigheter hvordan man skulle takle epidemien. Dette gjaldt også innen EU, siden helsepolitikk ikke er en del av traktaten. Forskjellige land, både i Europa og resten av verden, angrep pandemien på forskjellige måter. Dette var noe nytt, og en situasjon som beslutningsmessig var godt ute på den ustrukturerte enden av skalaen.

For Norges del var det tre ting som traff samtidig:

- Corona-viruset
- Massepermitteringer i reiselivsnæringer og andre servicenæringer
- Dramatisk fall i oljeprisen

De to første av disse punktene var felles for alle land som ble rammet. Det tredje kom som et tillegg for Norge og andre oljeproduserende land.

Myndighetene måtte ta beslutninger om hvordan samfunnet best skulle møte en sykdom der potensielt et stort antall mennesker ville kreve omfattende sykehusbehandling, og samtidig prøve å holde landets økonomi gående.

Et annet eksempel på en tilnærmet utstrukturert beslutningssituasjon er redningsaksjonen for Apollo 13 i 1970. De fleste har sikkert sett filmen med Tom Hanks i hovedrollen. Situasjonen var at en eksplosjon i en oksygentank satte romskipet ut av spill. Gjennom et intenst samarbeid mellom mannskapet og bakkekontrollen klarte de, mot alle odds, å få mannskapet velberget tilbake til Jorden.

Figur 3.5 Det virkelige mannskapet på Apollo 13 (Foto: NASA, Wikimedia Commons https://commons.wikimedia.org/wiki/File:Apollo_13_Prime_Crew.jpg)

Kapittel 4 Forretningsprosesser og styring

Forretningsprosesser står sentralt i dagens organisasjonsutvikling. Vi har tidligere sett på hvordan vi kan dele en organisasjon i funksjonsområder. Forretningsprosesser går gjerne på tvers av disse, noe som gjør det nødvendig å se prosessene som helheter for å kunne optimalisere dem. Dette gjør imidlertid at prosessene tradisjonelt ikke har hatt noen "eier" i organisasjonen. De vises ikke på organisasjonskartet, og har heller ikke fått navn. Samtidig er prosessene helt sentrale for virksomhetens konkurransekraft. Det er prosessene som utgjør virksomhetens kjernekompetanse. Prosessene skal sammen skape merverdi for kunden. Å tenke i prosesser er dermed et klart kundeorientert perspektiv.

4.1 Prosesser

En forretningsprosess er i følge Hammer og Champy (Hammer 1993) "en samling aktiviteter som tar et eller flere slags input og lager en output som er av verdi for kunden" (min oversettelse). Forretningsprosesser skal skape en merverdi for kunden (enkelt sagt det som gjør at kunden kjøper vårt produkt i stedet for å sette det sammen selv). En forretningsprosess består av flere aktiviteter, som utføres i en bestemt rekkefølge. Aktivitetene kan være spredt på forskjellige deler av organisasjonen.

Vi kan si at en prosess utgjør en måte å arbeide på. Om to bedrifter som lager samme produkt (output) av de samme delene (input), kan de utføre selve arbeidet på mange forskjellige måter. Noen måter å gjøre det på vil være mer effektive enn andre. Graden av effektivitet vil gjenspeiles blant annet i kostnader, men også i leveringstider, servicegrad med mer. Effektiviteten av forretningsprosessene har dermed direkte sammenheng med konkurranse-kraften.

Et historisk eksempel på en som forandret radikalt på interne forretningsprosesser, var Henry Ford. Da han gikk inn i bilbransjen, hadde bilene en slik pris at de bare var aktuelle for de rike. Hovedårsaken til dette var at bilene ble håndlaget. Det tok lang tid å sette sammen en bil. Henry Fords idé var å dele opp monteringsprosessen i klart definerte enkeltoppgaver, som så ble utført av hver sin arbeider. For å få til dette, innførte han samlebåndet. Arbeiderne stod langs et rullebånd, der de utførte sin spesifikke oppgave etter som bilene rullet forbi. Dette gjorde det mulig å produsere biler (T-Ford) til en pris som gjorde at et stort antall amerikanere hadde råd til dem.

På 1990-tallet kom forretningsprosessene i fokus for alvor. Metoder som Business Process Reengineering (BPR) hadde sin storhetstid på midten av tiåret. Den grunnleggende tanken var at man ved å ta i bruk informasjonsteknologi kunne utføre prosessene på radikalt nye måter.

Vi skiller gjerne mellom tre typer prosesser:

- Ledelsesprosesser: Prosesser som styrer organisasjonen som helhet. Et eksempel på en slik prosess er strategiledelse.
- Operative prosesser: Dette er de prosessene som utgjør kjernevirksomheten og bidrar til organisasjonens produkter. Eksempler er bestilling, produksjon og salg.

- Støtteprosesser: Prosesser som støtter de operative prosessene. Typiske prosesser er regnskap og rekruttering.

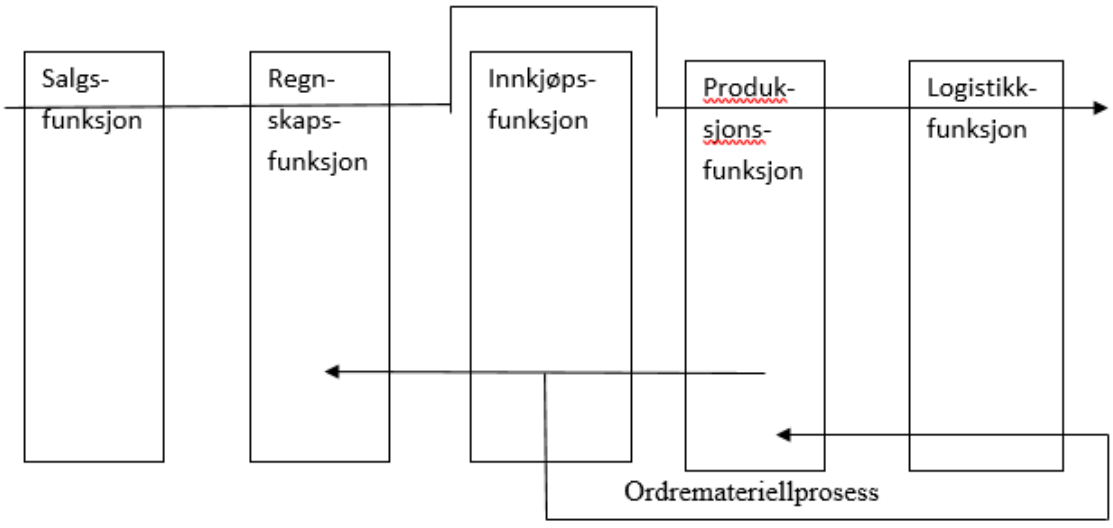
For bedriften gjelder det å bryte opp prosessene i aktiviteter, og ordne dem i rekkefølge. Videre er det viktig å måle hvor effektive disse oppgavene utføres. Ofte vil man også definere et nivå med subprosesser under hovedprosessen.

Et eksempel på hvordan forretningsprosesser omfatter flere funksjonsområder er beskrevet av Monk og Wagner (Monk 2007). Eksempelet gjelder salg av datamaskiner. Bedriften som selger disse, tilbyr også finansiering av maskinene. Maskinene skal selvsagt leveres, og dessuten tilbyr de teknisk støtte etter salget. I salget av en datamaskin inngår dermed flere prosesser, som kan fremstilles i en tabell.

Input	**Funksjonsområde ansvarlig for input**	**Prosess**	**Output**
Forespørsel om å kjøpe datamaskin	Salg/markedsføring	Ordre	Ordre laget
Finansiell hjelp til kjøp	Regnskap og finans	Ordne finansiering internt	Kundens finansiering gjennom selskapet
Teknisk støtte	Salg/markedsføring	Hjelpelinje tilgjengelig	Kundens tekniske problemer løst
Komplettering av ordre	Forsyningskjedeledelse	Forsendelse og levering	Kunden mottar levering

Tabell 4.1 Prosesser involvert i kjøp (etter Monk 2007)

Prosessene viser kundenes perspektiv på virksomheten. Prosessene går typisk på tvers av funksjonene i virksomheten. Når vi ser på en kundes ordre, omfatter denne disse funksjonene (ibid.):



Figur 4.3 Prosessperspektiv på virksomheten (etter Monk 2007)

Kunden skal ikke behøve forholde seg til de forskjellige funksjonsområdene i virksomheten. Informasjonssystemene skal være slik at kunden forholder seg til det som er output fra prosessene. Dette er en grunnleggende idé bak ERP-systemer. Figur 5.3

viser hvordan en ordre fra en kunde krysser flere funksjonsområder i virksomheten. Et *integrert informasjonssystem* (ERP) skal sørge for at dette skjer automatisk.

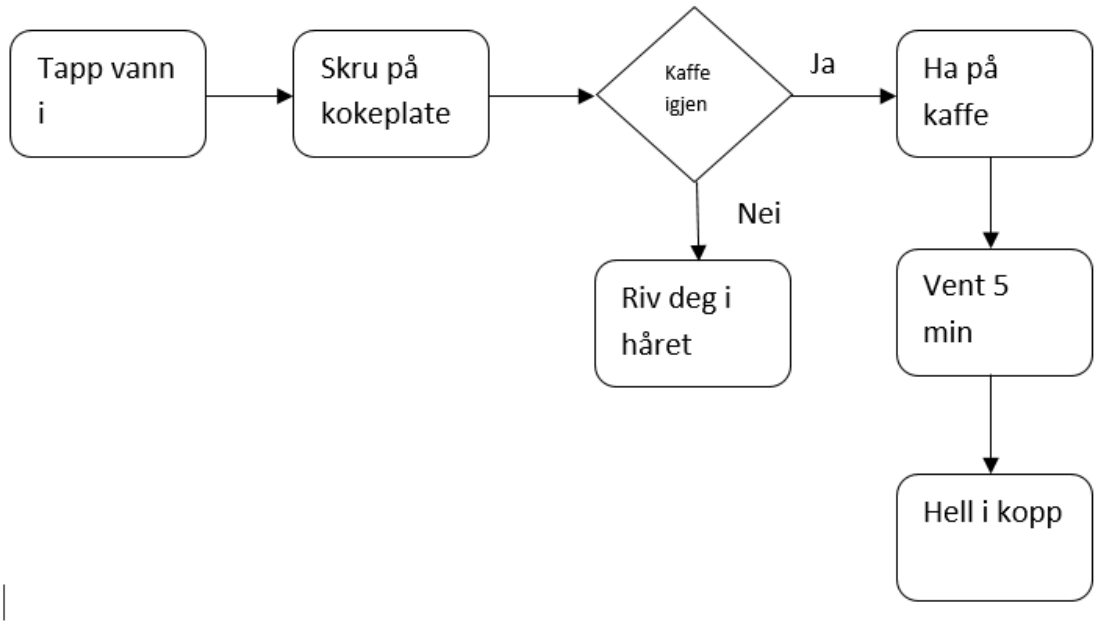
4.2 Beskrivelse av prosesser

En prosess kan beskrives verbalt, som denne:

Prosess: Koke kaffe

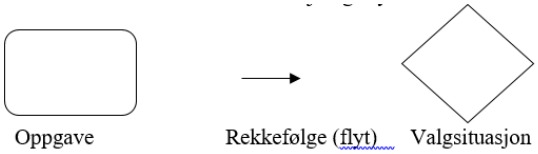
- Tapp vann i kaffekannen
- Skru på kokeplaten
- Er det kaffe igjen?
    - Hvis ja: Ha på kaffe når vannet koker
    - Vent 5 minutter
    - Hell kaffe i kopp
- Hvis ikke: riv deg i håret

Her er det en beslutningssituasjon i prosessen (er det kaffe igjen?). I en prosess kan det være mange slike, med forskjellig rekke av oppgaver avhengig av utfallet av beslutningen. En verbal beskrivelse kan derfor fort bli uoversiktlig. Dette er grunnen til at man gjerne bruker forskjellige beskrivelsesteknikker for å beskrive hva som skjer i prosessen. Kaffekokingen kan vi for eksempel beskrive slik:



Figur 4.3 Flytdiagram for kaffekoking

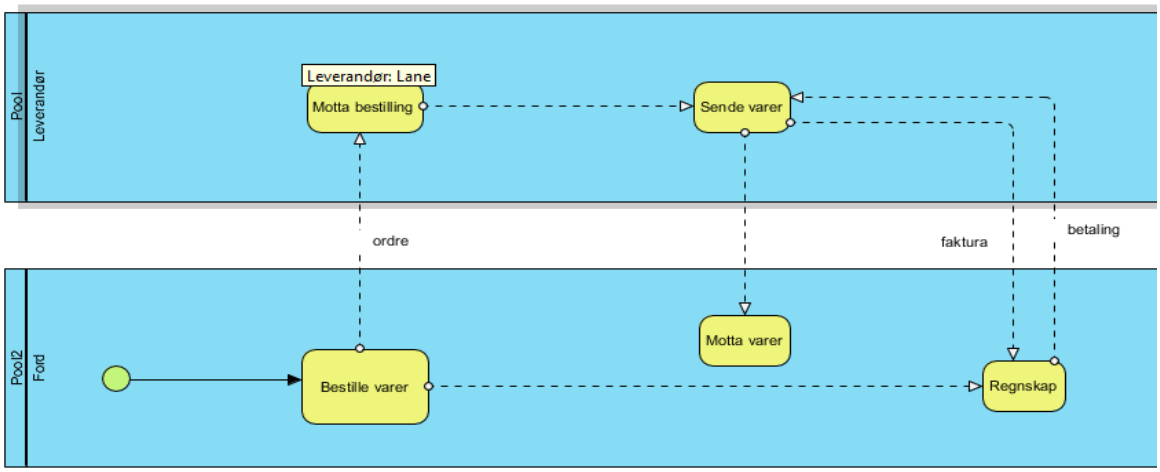
Vi ser her at det er brukt tre forskjellige symboler:



Figur 4.4 Symboler brukt i prosessdiagram

På denne måten kan man bygge opp detaljerte beskrivelser av kompliserte prosesser. I Business Process management arbeider man med dokumentasjon av prosesser, måling av effektiviteten og kontinuerlig forbedring.

Beskrivelser kan godt være mindre formelle enn dette. For presentasjon for personer som ikke er trent i prosessdiagrammer, kan man gjerne bruke mer visuelle beskrivelser. Nedenfor er en beskrivelse av fakturamottaket i Ford Motor Company i USA før den ble lagt om (Sethi 1998):



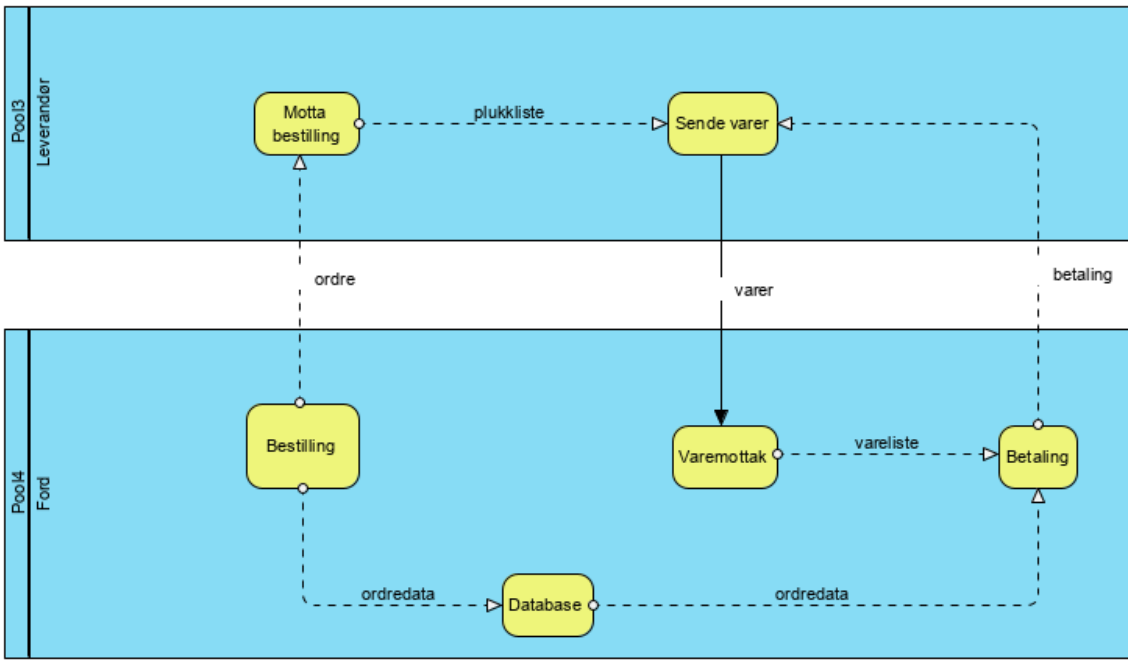
Figur 4.5 Tidligere fakturabehandling hos Ford USA(etter Sethi 1998)

Dette er en klassiker innen litteraturen om Business Processing Reengineering. I begynnelsen av 1980-årene hadde Ford USA 500 personer som arbeidet med fakturabehandling (fakturaer fra leverandørene for det Ford hadde kjøpt fra disse). Målet for ledelsen var å få antallet som arbeidet med fakturabehandling ned til 400. Da de oppdaget at konkurrenten Mazda klarte seg med bare 5 personer til samme oppgave, innså de at det måtte være mulig med klart mer dramatiske omleggginger.

Opprinnelig var planen å gjennomføre tradisjonell rasjonalisering, med støtte fra datamaskiner. Nå ble målet å forandre selve arbeidsprosessen.

Den gamle måten å arbeide på fremgår av figur 4.5. Først ble det skrevet en bestilling, med kopi til fakturabehandling. Når varene så ble mottatt, ble fraktdokumentene sendt til fakturabehandling. I mellomtiden sendte leverandøren en faktura. Denne ble så sammenlignet med kopien av den opprinnelige bestillingen og fraktdokumentene. Hvis alt var i orden, ble fakturaen betalt. Nå viste det seg at avdelingen for fakturabehandling brukte mesteparten av tiden på tilfelle der det var avvik mellom dokumentene. Avviket måtte da undersøkes, noe som tok sin tid.

Med den nye prosedyren, som er vist i figur 4.6, søkte man å unngå avvik overhode. Ford innførte fakturaløse innkjøp. Bestillingene ble lagt i en database, og når varene mottas, kontrolleres leveransen mot bestillinger i databasen. Hvis de matcher en bestilling, aksepteres leveransen. Informasjonssystemet sjekker automatisk leveransen mot bestillingen, og overfører så betalingen for dem. Leverandørene sender aldri noen faktura.



Figur 4.6 Omdesignet fakturabehandling hos Ford (ibid.)

Eksempelet viser et sentralt prinsipp i design av forretningsprosesser: unngå dødtid. Manuell kontroll av fakturaer og undersøkelse av avvik er dødtid som kunden ikke skal betale for. Kunden betaler for et produkt, og er ikke interessert i å betale mer for manglende effektivitet internt.

Beskrivelse av prosesser er noe man har drevet med i mange tiår innen ingeniørfag, der man arbeider med å konstruere maskiner for å utføre prosesser. Innen ingeniørfagene finnes det standard symboler for prosessbeskrivelser. Typiske symboler for programmerere er vist nedenfor.

4.3 Business Process Modeling Notation

Ingeniørenes og programmerernes symboler for flytdiagrammer er ikke spesielt egnet for å beskrive prosesser i organisasjoner. For å rydde opp i de mange individuelle måtene å tegne flytdiagrammer på, ble en standard foreslått av en organisasjon som kaller seg Business Process Management Initiative. Denne standarden har fått navnet Business Process Modeling Notation, forkortet BPMN. Vi skal her se på versjon 1 av denne.

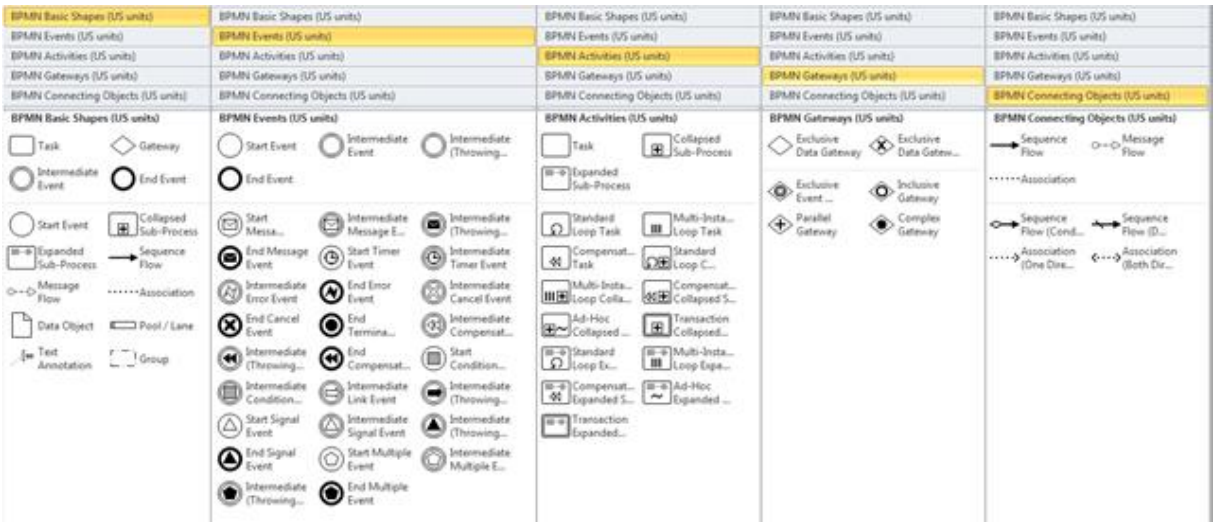
I BPMN er det fire grunnleggende kategorier av symboler:

- Flytobjekter (Flow Objects)
- Forbindelsesobjekter (Connecting Objects)
- Svømmebaner (Swimlanes)
- Artefakter (Artifacts)

Innenfor disse fire kategoriene finner vi selve symbolene, slik at vi får denne komplette listen:

- Flytobjekter:
  - Hendelser (Events)
  - Aktiviteter (Activities)
  - Porter (Gateways)
- Forbindelsesoblekter:
  - Sekvensflyt (Sequence Flow)
  - Meldingsflyt (Message Flow)
  - Sammenheng (Association)
- Svømmebaner;
  - Basseng (Pool)
  - Bane (Lane)
- Artefakter:
  - Dataobjekter (Data Objects)
  - Gruppe (Group)
  - Merknad (Annotation)

Vi skal se nærmere på disse symbolene og hvordan de brukes. Microsoft Visio støtter for øvrig BPMN, og viser følgende palett med symboler:



Figur 4.7 BPMN-symboler i Microsoft Visio

De grunnleggende symbolene er de vi finner i kategorien Flytobjekter. Av disse er det hendelser som representerer start og slutt på en prosess. Disse kan bare ha piler ut fra seg (start) og inn (slutt), og representeres med forskjellige symboler. I tillegg kan vi ha mellomhendelser (Intermediate Events), som kan ha piler både inn og ut. Disse tre typene hendelser har hver sine symboler:

Figur 4.8 Hendelsessymboler i BPMN

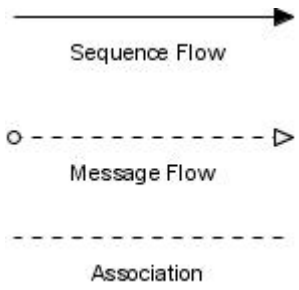
Aktiviteter representerer oppgavene som skal gjøres, og beskrives med avrundede rektangler:

Figur 4.9 Aktivitetssymboler i BPMN (ibid.)

Portene representerer forgreninger eller sammenføyninger i prosessløpet. Symbolene som brukes er disse:

Figur 4.10 Portsymboler (ibid.)

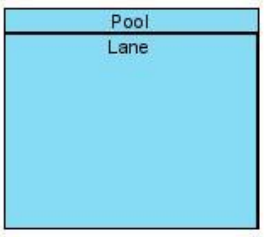
De grunnleggende symbolene knyttes sammen med flytobjekter. Disse kan være av tre typer: sekvenser, meldinger og sammenhenger. Symbolene er disse:



Figur 4.11 Flytobjekter (ibid.)

Prosessmodellene med sine elementer organiseres i enheter som assosieres med svømmebassender. Modellen som helhet ligger innenfor et basseng (Pool). Innenfor et basseng kan vi ha flere parallelle løp. Disse kalles for svømmebaner (Swimlanes), akkurat som et basseng for konkurransesvømming er delt opp i løp:

I BPMN er det et symbol kan representere hele samlingen, dvs bassenget, og også vise svømmebanene. Et basseng kan inneholde en eller flere baner. Disse kan vises i diagrammet, eller man kan vise bare helheten, dvs bassenget som et rektangel uten detaljer.



Figur 4.13 Swimlane-symbolet i BPMN

Vi har nå presentert det meste av "verktøykassen" for å lage flytdiagrammer med BPMN. Nedenfor er vist et ferdig diagram som viser behandling av elektronisk stemmegivning.

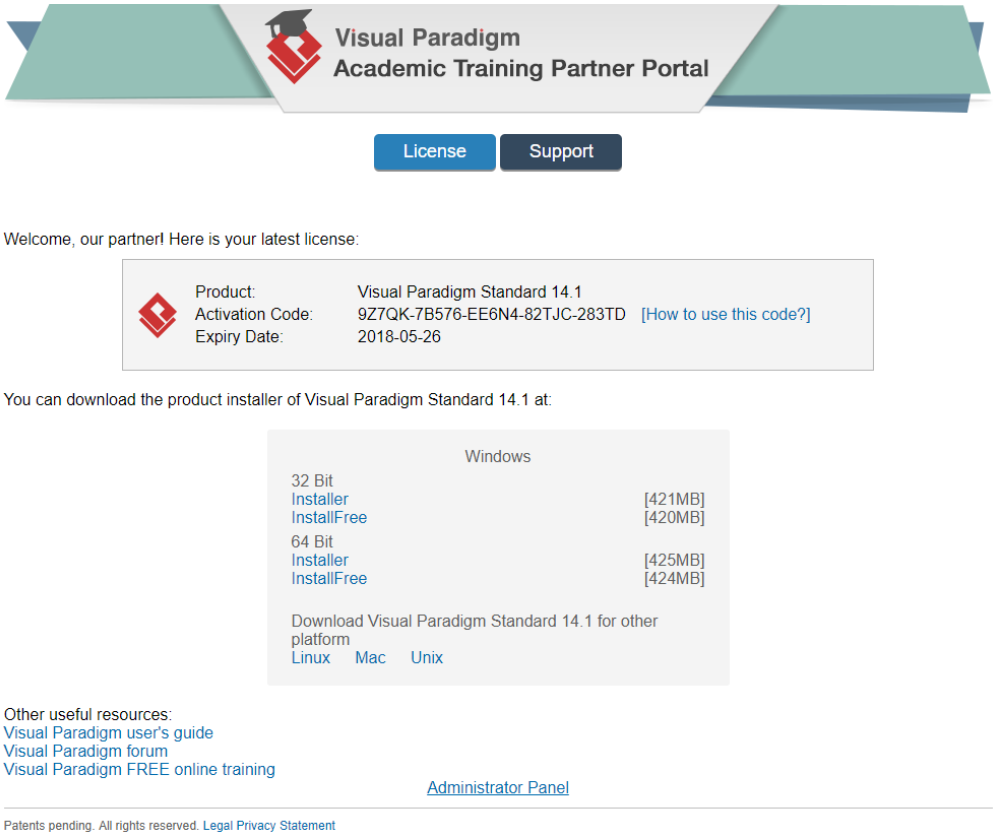
4.4 Visual Paradigm – et verktøy for modellering

Visual Paradigm er et av de mest brukte modelleringsverktøyene. Det brukes bl. a. av Kartverket. Visual Paradigm er skrevet i Java og kan kjøres på alle aktuelle plattformer.

Nedlasting

For studenter ved USN: Gå til denne nettsiden: https://ap.visual-paradigm.com/buskerud-and-vestfold-university-college

Siden ser slik ut:



Som man ser, kan man her laste ned installasjonsfil for Windows, Linux, Unix og Mac. Her finnes instrukser for installasjonen:

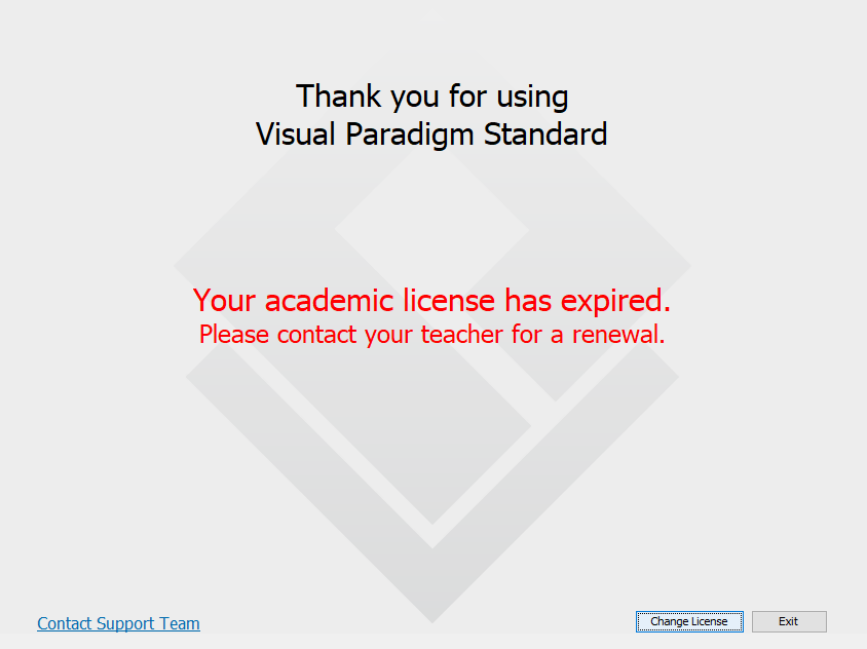
Windows: https://www.visual-paradigm.com/support/documents/vpuserguide/12/14/6008_windows2000n.html

MacOS: https://www.visual-paradigm.com/support/documents/vpuserguide/12/14/6042_macosx.html

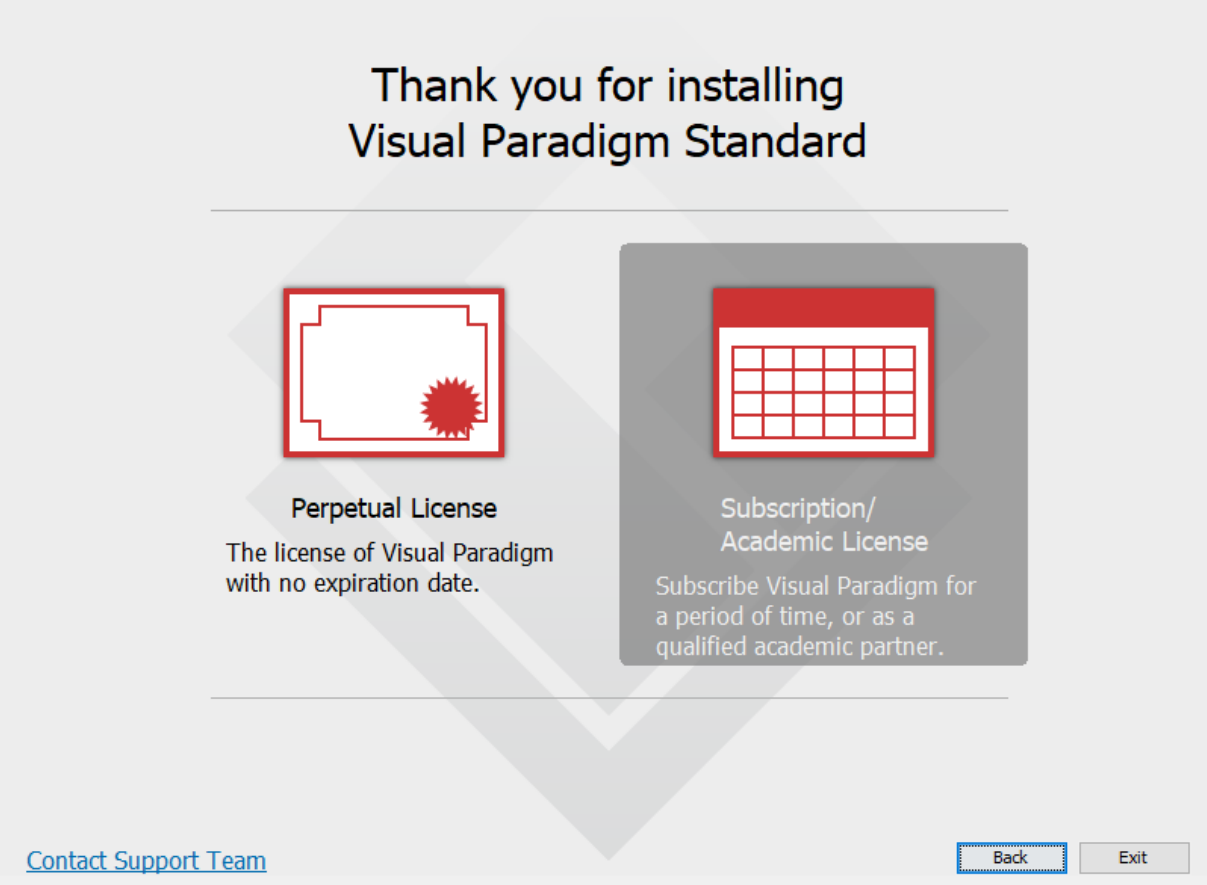
Linux: https://www.visual-paradigm.com/support/documents/vpuserguide/12/14/6043_linuxandunix.html

Oppstart første gang

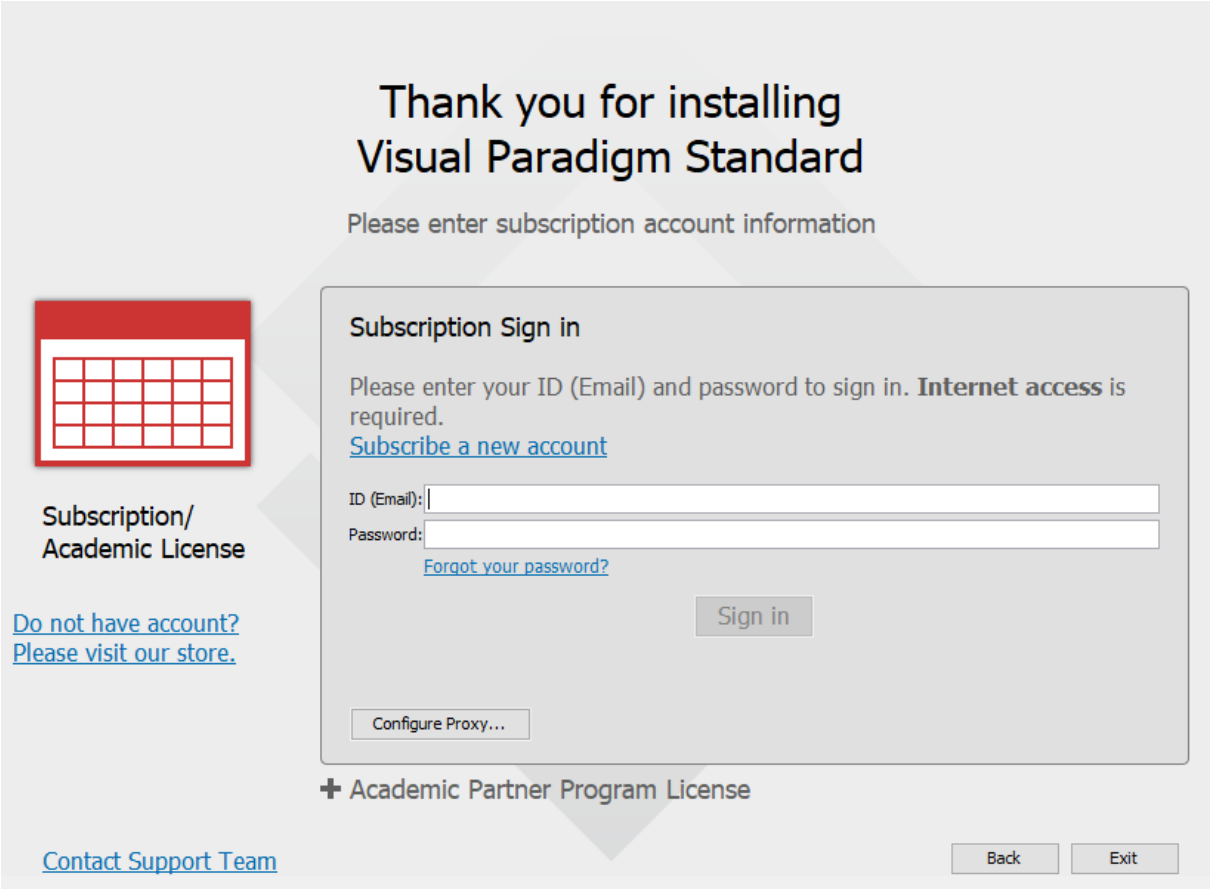
Når dere starter Visual Paradigm første gang, får dere beskjed om at lisensen er utløpt:



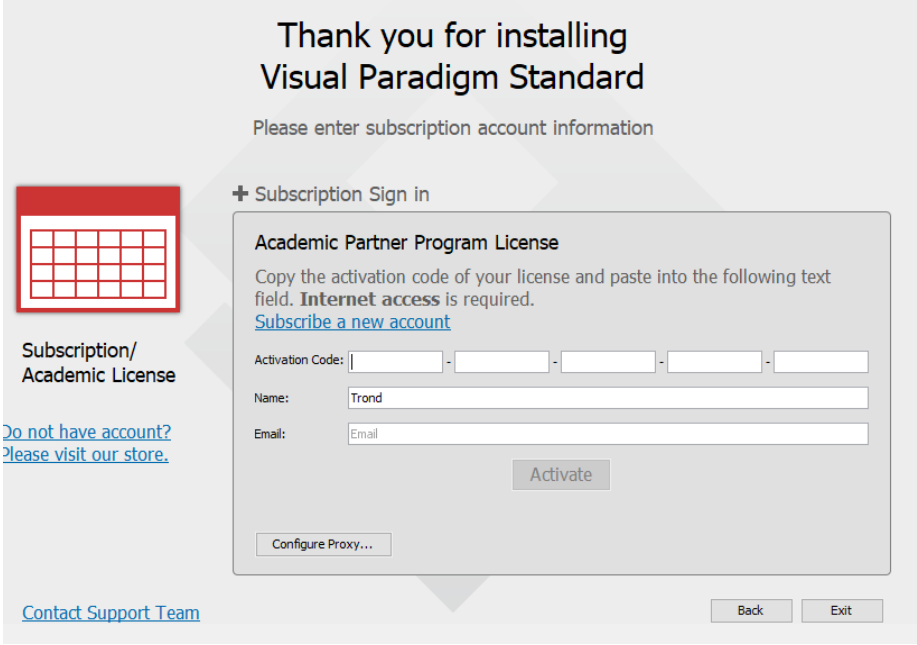
Klikk på Change licence, og dere får opp følgende skjermbilde:



Velg Subscription/Academic Licence, og dere får opp følgende side:

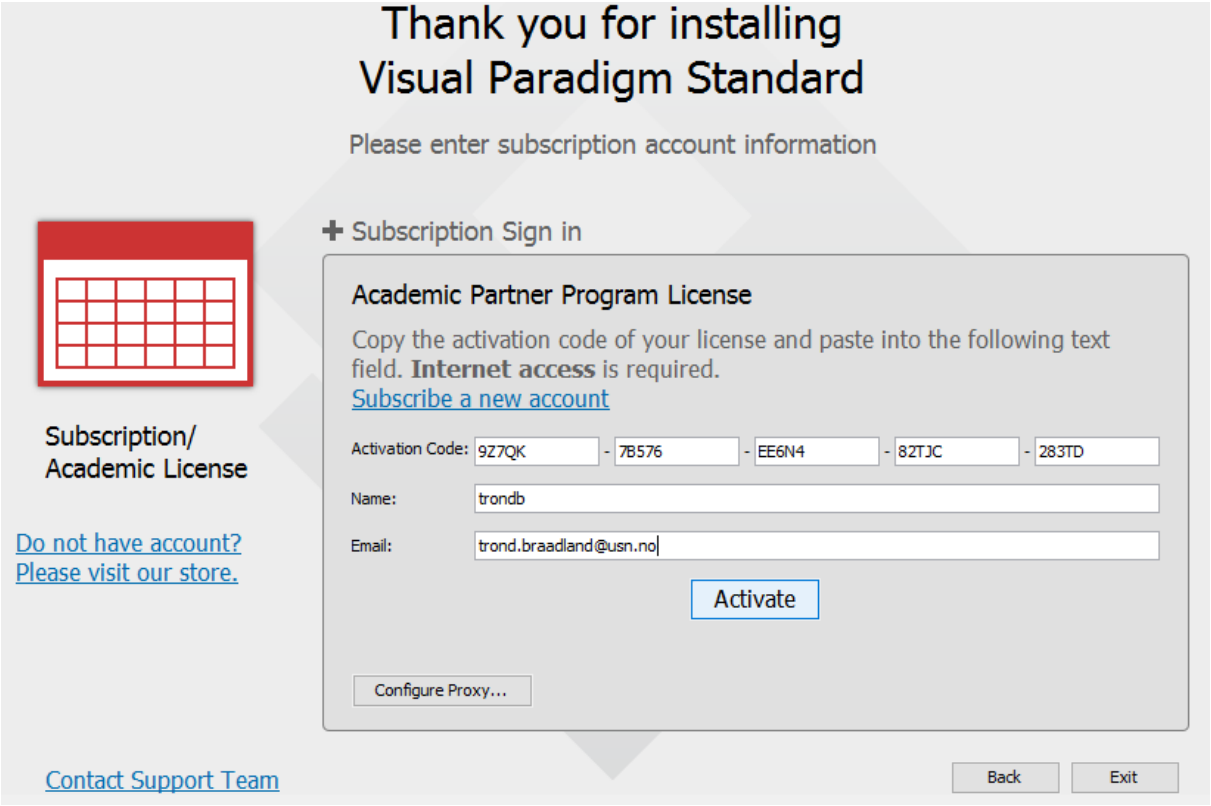


Klikk på Academic Partner Program Licence nederst i vinduet. Dere får opp dette vinduet:

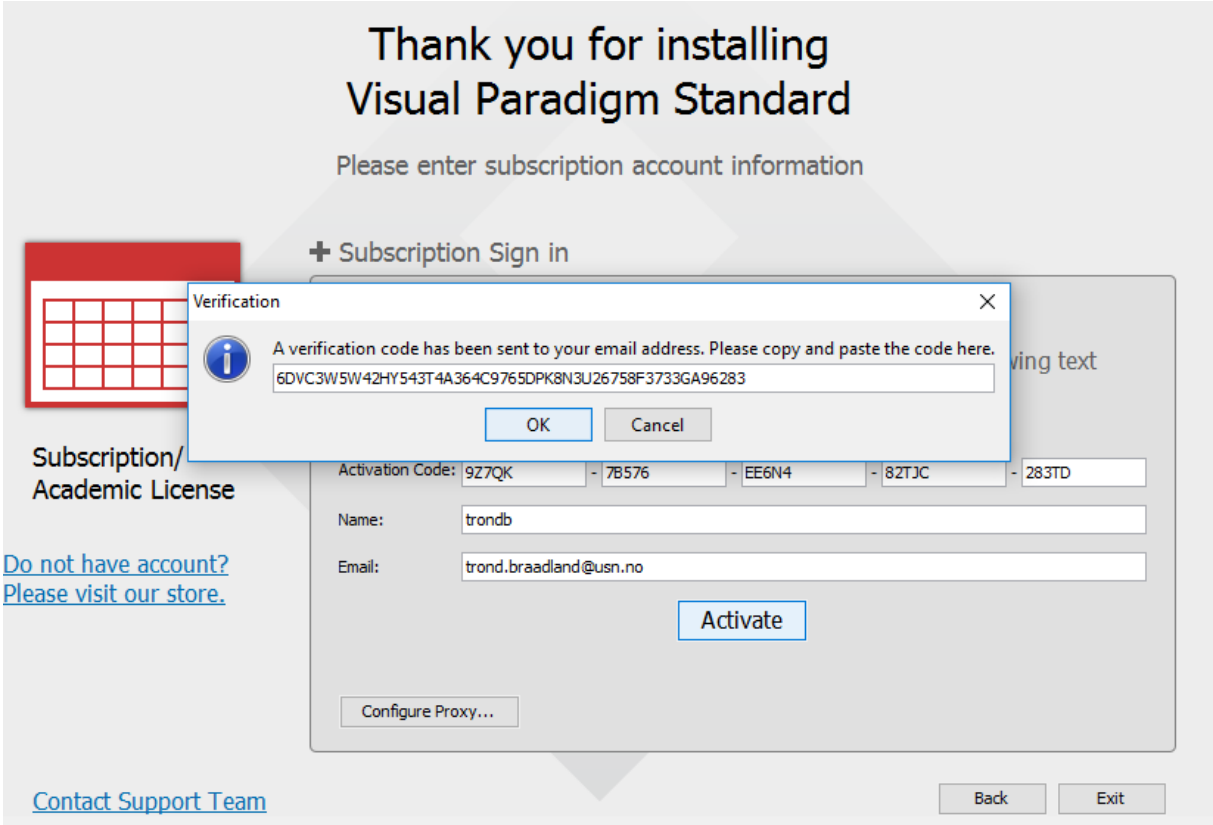


USN har en akademisk lisensavtale med Visual Paradigm. Det betyr at dere får full tilgang til programmet i ett år. Aktiveringskode finner dere her: https://ap.visual-paradigm.com/buskerud-and-vestfold-university-college

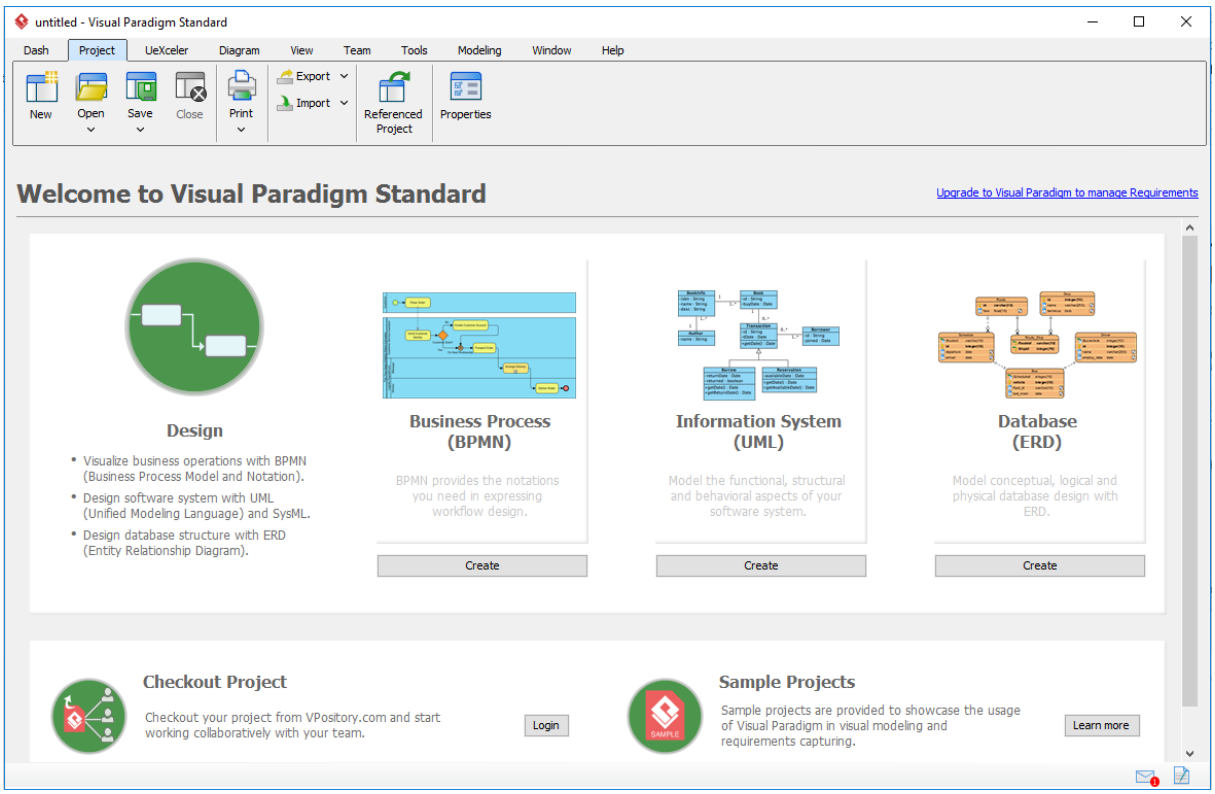
Dere kopierer så inn lisensen (bruk Ctrl V):



Deretter får dere tilsendt en ny kode til den adressen dere har oppgitt. Et vindu for å kopiere inn denne dukker opp. Dere må paste ved å bruke Ctrl V:



Nå er VP aktivert, og dere kan begynne. Åpningsvinduet ser slik ut:



Denne siden heter Start Page, og den ligger som et valg i Windows-menyen.

Diagramtyper

Som dere ser av startsiden, er det tre typer diagrammer vi kan lage med Visual Paradigm:

- Forretningsprosesser (kommer i kurset Business Intelligence og datavarehus)
- UML-diagrammer (alle typer)
- Datamodeller

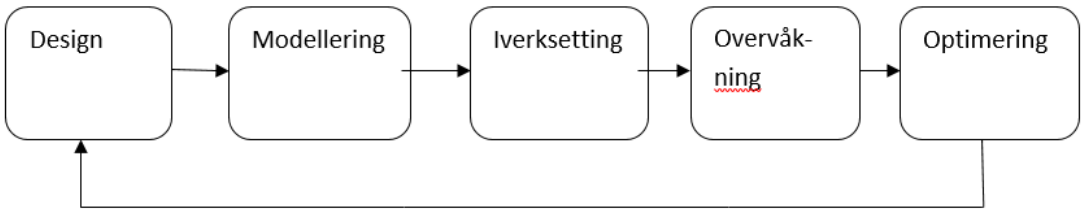
Vi skal i dette kurset bruke Visual Paradigm til å tegne UML-diagrammer.

4.5 Business Process Management

Business Process Management er en tilnærming til ledelse der fokus er på å oppfylle kundenes behov gjennom virksomhetens prosesser. Informasjonsteknologi er sentralt i prosessledelse fordi et stort antall oppgaver enten kan utføres av datamaskiner eller støttes av disse. I ERP-systemer er sentrale prosesser ferdig definert, og rekkefølgen mellom oppgavene er automatisert.

Virksomhetens prosesseledelse utgjør en livssyklus. Siden verden rundt bedriften stadig forandrer seg, må også prosessene både optimeres og forandres over tid. Det er også sentralt at man hele tiden kan måle effektiviteten av prosessene. Måling av effektivitet inngår i de interne målingene i et Business Intelligence-system.

Vi kan vise prosessledelsens livssyklus som et prosessdiagram:



Figur 4.17 Prosessledelsens livssyklus

Design: beskrive eksisterende prosesser og utforme nye.

Modellering: dette vil være en operasjonalisering av designen. Her inngår også analyser av typen what-if, for å undersøke effekten av forskjellig bruk av innsatsfaktorer.

Iverksetting: her tar man eventuelt i bruk programvare for å automatisere eller støtte prosessutførelsen, og definerer menneskers oppgaver i utførelsen.

Overvåkning: tilstanden til og ytelsen til prosessene måles. Dette skal avdekke flaskehalser eller ineffektive deler av prosessene.

Optimering: ut fra målinger av ytelser kan prosesser forbedres.

Business Process Management skal selvsagt være forankret i virksomhetens mål og strategier. Det bringer oss over i neste tema: balansert målstyring.

Kapittel 5 Datagrunnlaget: TPS og ERP

I dette kapittelet skal vi se nærmere på de systemene som støtter den daglige, operative virksomheten. Slike systemer kalles med forskjellige navn, vanlig i dagligtale er ”smør og brød”-systemer eller driftssystemer. Den faglige betegnelsen er imidlertid transaksjonsprosesseringssystemer, forkortet TPS. Vi skal også se nærmere på de moderne, integrerte systemene som gjerne kalles Enterprise Resorce Systems eller Enterprise Resorce Planning Systems, forkortet ERP. På norsk bruker vi betegnelsen foretakssystemer.

5.1 Brukernes møte med informasjonssystemene

Som kunder eller brukere er vi daglig i kontakt med informasjonssystemer. Det vi møter, er det som ofte kalles front-end systemer. På engelsk brukes Point of Sales – salgspunkt. Gjennom disse gir og mottar vi informasjons fra de bakenforliggende systemene. Vi skal se på noen eksempler.

I butikkene ekspederes vi ved hjelp av elektroniske kasser. En *scanner* brukes til å lese strekkoden på varene. Prisene hentes frem, og kassen regner ut totalsum. Det er viktig å være klar over at all informasjon om kjøpet sp sendes til det bakenforliggende butikkdatasystemet, der de brukes til en rekke forskjellige rapporter og analyser. En elektronisk betalingsterminal hører gjerne også til kassen.

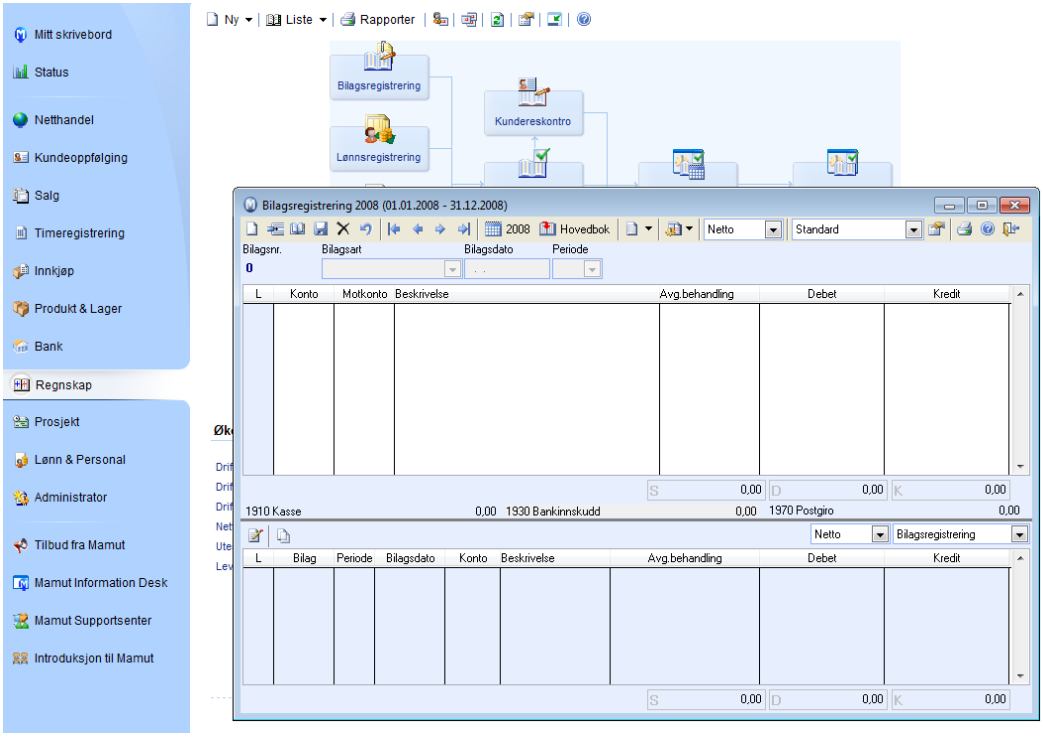
Bankenes bankautomater har gjort kontantuttak tilgjengelige døgnet rundt hele uken. Igjen står automaten i forbindelse med bankens sentrale systemer, der alle kontoopplysninger er lagret.

Bruk av nettbutikker er etter hvert blitt så omfattende at de truer de gamle, fysiske butikkene. Nær sagt hva som helst kan i dag kjøpes i nettbutikker. Bak nettbutikken ligger informasjonssystemer av samme type som butikkdatasystemer, men med tillegg som bilder og omtaler av varene. Kundene kan i mange nettbutikker også lage ønskelister som lagres.

Bruk av nettbanker er etter hvert blitt svært utbredt. Nettbanken kommuniserer med det samme bakenforliggende systemet som minibankene.

Også reiser er det nå vanlig å bestille på nettet. Informasjon om avganger og priser ligger i et bakenforliggende datasystem, og alle opplysninger om kjøpet av reisen lagres også der.

Andre systemer arbeider vi med på jobben, som regnskapssystemet vist nedenfor.



Figur 5.1 Føring av regnskap i Mamut

Alle disse systemene, og de utallige andre som finnes, vil senere utgjøre kilden til den informasjonen vi trenger for *forretningsanalyse* - Business Intelligence.

5.2 Transaksjonsprosesseringssystemer

I den daglige driften av en virksomhet er det en rekke repetitive oppgaver som må utføres. Slike oppgaver kan være innkjøp, fakturering, registrering av timelister, beregning av lønn, oppdatering av kontoer, registrering av bilag m.m. Vi kaller slike oppgaver transaksjoner. Mer nøyaktig er en transaksjon en forretningshendelse som skaper eller modifiserer lagrede data i et informasjonssystem (Alter2002). Typisk for transaksjoner er at de utføres etter nøye fastlagte regler, noe som igjen betyr at de kan automatiseres.

I en virksomhet hører transaksjonsprosesseringssystemene hjemme i den daglige driften. Dette gjelder kontosystemer, logistikksystemer, billettreservasjonssystemer, produksjonsplanleggingssystemer samt mye mer. I tillegg kommer en støttefunksjon som regnskap.

Bruken av TPS'er skal effektivisere virksomheten. I en moderne bedrift av noen størrelse utføres et svært stort antall transaksjoner hver dag, og det ville både være kostbart og håpløst langsomt å utføre disse manuelt. Ofte er også selve transaksjonen ganske komplisert, og den må utføres med null feiltoleranse.

Et TPS mottar inndata fra en eller flere kilder (manuell inntasting, lesing av strekkoder, datafiler m.m.), behandler dem og leverer de behandlede utdata fra seg i form av skjermpresentasjoner eller dokumenter. Det er to måter selve behandlingen kan skje på:

- Online, som vil si at brukeren er i interaksjon med systemet. Brukeren får øyeblikkelig tilbakemelding om både hva som skal gjøres og eventuelle feil, og også utdata kan presenteres øyeblikkelig, forutsatt at datamengdene ikke er for store. Uttak av penger i en minibank, eller betaling med en kortleser i butikken, er eksempler på dette. TPS'er som hovedsakelig brukes interaktivt, kalles også online transaksjonsprosesseringssystemer, eller OLTPS, og selve behandlingen online transaksjonsbehandling eller bare OLTP
- Batch eller satsvis behandling: med store datavolumer og/eller komplisert behandling er det ofte lite hensiktsmessig med online behandling. Det kan for mange oppgavers vedkommende ta timer før systemet er ferdig. Eksempler på dette er bankenes oppdateringer av kontoer mot andre banker, som typisk kjører om natten, lønnsberegning for store virksomheter, eller gjennomgang av lagersystemer for å få en oversikt over hvilke varer som må bestilles. Slike oppgaver settes i gang som en jobb, der et eget jobbkontrollprogram styrer utføringen. Slike jobber startes vanligvis automatisk. Dette er typisk for stormaskiner og deres operativsystemer.

Transaksjonsprosesseringssystemenes database inneholder en blanding av flyktige og mer varige data. For eksempel vil saldoen på en bankkonto være flyktig, idet den til stadighet bli oppdatert, mens data om kontohaveren er mer varige (i dette tilfellet vil det si at de sjelden oppdateres). Andre data oppdateres aldri, som selve transaksjonen. Når en transaksjon er utført, skal den ikke kunne oppdateres etterpå.

For mange TPS'er er det helt kritisk at de er i drift kontinuerlig. For både banken og kundene er det katastrofalt om de sentrale TPS'ene skulle være ute av drift noen timer. Både operativsystemene og databasehåndteringssystemene som brukes har en rekke egenskaper somskal sikre kontinuerlig drift. Dette gjelder også lagringen av data. Disse store systemene bruker egne platelagerenheter bygget for rask aksess og feilfri drift (såkalte RAID's), og ofte brukes også speiling av data, dvs at de samtidig ligger på to uavhengige platelagre. Backup og recovery er også sentrale elementer.

Både det at systemene må være kontinuerlig tilgjengelige, og at databasen består av kanskje hundrevis av tabeller, gjør at TPS'er ikke er egnet til rapportering. De fleste slike systemer har noen enkle rapportmuligheter, som regel i form av standardiserte rapporter, ferdig programmert slik at man vet hva de gjør og hvor mye maskinressurser de vil bruke. Å sette i gang en komplisert SQL-spørring på et slikt system er ofte utelukket, idet man ikke kan vite sikkert hvor mye den vil kreve. For ledelsen er imidlertid ikke slike standardrapporter til særlig hjelp annet enn til rutinemessige dag-for-dag beslutninger.

5.3 Om det tekniske

Begrepet transaksjon har ulike betydninger i forskjellige sammenhenger. I økonomifaget brukes transaksjon om en kjøp/salgs-situasjon, og i organisasjonsfaget er transaksjonskostteori en av de nyere organisasjonsteorier. I forbindelse med informasjonsteknologi har imidlertid transaksjon en annen betydning. Her står det for en

oppgave som består av flere mindre deler. Disse delene skal utføres i en bestemt rekkefølge, og alle må være fullført for at transaksjonen skal være utført.

Et eksempel: vi skal betale en regning på 1000 kroner i nettbanken. Dette er en enkelt transaksjon for banken, men den består egentlig av flere aktiviteter. La oss forenkle det hele til bare to aktiviteter:

- Betalerens konto skal debiteres
- Mottagerens konto skal krediteres

Hvis bare debiteringen fullføres, vil 1000 kroner ha "blitt borte" på veien. For at transaksjonen skal være utført, må både debitering og kreditering være fullført. Hvis noe galt skulle skje etter debitering, slik at transaksjonen ikke kan fullføres, vil et TPS tilbakestille det hele slik at ikke noe galt er skjedd.

I virkeligheten vil både debitering og kreditering bestå av flere aktiviteter, som å sjekke at det er riktige kontoer, at det er penger nok på betalerens konto m.m.

Et TPS er imidlertid ikke laget for å utføre en transaksjon om gangen. Disse systemene kan behandle et stort antall transaksjoner samtidig, fra mange forskjellige brukere. Dette krever avanserte teknologiske løsninger:

- Tidsdeling, der prosessorens tid deles mellom brukerne og transaksjonene. Bare en oppgave kan ligge i prosessoren i et gitt øyeblikk, men prosessoren veksler så raskt mellom oppgavene at det for brukerne virker som hver enkelt av dem bruker systemet alene.
- Låsing av data. Et TPS skal sørge for at forskjellige transaksjoner ikke skal kunne bruke de samme dataene samtidig.

Operativsystemer som Windows og Mac OS er ikke laget for å håndtere transaksjoner. Typiske operativsystemer for et TPS er UNIX og IBM z/OS.

5.4 Typer av transaksjonsprosesseringssystemer

Vi kan dele transaksjonsprosesseringssystemene i flere kategorier ut fra hvilken funksjon de støtter i virksomheten. Den vanlige inndelingen er etter de fem forskjellige funksjonsområdene (Laudon 2002):

- Finans og økonomi
- Menneskelige ressurser
- Markedsføring og salg
- Material- og produksjonsstyring
- Annet

Ut fra denne inndelingen får vi fem kategorier av TPS'er:

- Systemer for finans og økonomi: det mest typiske her er regnskaps- og økonomistyringssystemer, som brukes til registrering av alle virksomhetens kostnader og inntekter.
- Systemer for menneskelige ressurser: et typisk eksempel er systemer for beregning av lønn til ansatte
- Systemer for markedsføring og salg: her finner vi en rekke forskjellige systemer, som salgsstøttesystemer (disse holder rede på alle kontakter mellom selger og kunde) og butikkdatasystemer.
- Systemer for material- og produksjonsstyring: her finner vi både lagerstyringssystemer og spesialiserte systemer for planlegging av produksjon i fabrikker. Disse systemene kalles MRP-systemer, for Materials Resource Planning
- Andre systemer: dette er en sekkepost for forskjellige typer bransjespesifikke systemer. En høyskoles studieadministrative system kan være et eksempel på et slikt.

Mange store bedrifter sitter på en stor portefølje av forskjellige transaksjonsprosesseringssystemer. Disse vil typisk være utviklet over mange år, og kan ofte kjøre på forskjellige tekniske plattformer. Slike systemer kalles på engelsk legacy systems. Ofte er det teknisk vanskelig å koble disse systemene mot hverandre. Spesielt for å få relevant informasjon for beslutningstagning, er det relevant å samle data fra forskjellige systemer.

I noen bedrifter finner vi en sentral avdeling for data drift, i andre har forskjellige organisatoriske avdelinger ansvar for sine systemer. I dag er det mange virksomheter som har outsorcet data drift, det vil si at en ekstern datasentral står for driften av systemene.

5.5 Legacy Systems

Legacy System betyr noe slikt som «nedarvede systemer». Det brukes om systemer som bygger på det som nå er gammel teknologi, enten det er maskinvare, operativsystem, programmeringsspråk eller databasesystem (ofte alle disse i kombinasjon). Mange virksomheter har slike systemer. Årsaken er som oftest at de fortsatt virker bra. Det kan også være at kostnaden ved å bytte dem ut er så høy at det ikke er prioritert. Et problem med disse systemene er at de ofte er dårlig dokumentert.

Wikipedia (https://en.wikipedia.org/wiki/Legacy_system) lister opp disse årsakene til at virksomheter fortsetter å bruke slike systemer:

- The system works satisfactorily, and the owner sees no reason to change it.
- The costs of redesigning or replacing the system are prohibitive because it is large, monolithic, and/or complex.
- Retraining on a new system would be costly in lost time and money, compared to the anticipated appreciable benefits of replacing it (which may be zero).
- The system requires near-constant availability, so it cannot be taken out of service, and the cost of designing a new system with a similar availability level is high.

Examples include systems to handle customers' accounts in banks, computer reservations systems, air traffic control, energy distribution (power grids), nuclear power plants, military defense installations, and systems such as the TOPS database.

- The way that the system works is not well understood. Such a situation can occur when the designers of the system have left the organization, and the system has either not been fully documented or documentation has been lost.
- The user expects that the system can easily be replaced when this becomes necessary.
- Newer systems perform undesirable (especially for individual or non-institutional users) secondary functions such as a) tracking and reporting of user activity and/or b) automatic updating that creates "back-door" security vulnerabilities and leaves end users dependent on the good faith and honesty of the vendor providing the updates. This problem is especially acute when these secondary functions of a newer system cannot be disabled.

Det er samtidig viktig å være klar over hvilke problemer disse systemene kan forårsake:

- If legacy software runs on only antiquated hardware, the cost of maintaining the system may eventually outweigh the cost of replacing both the software and hardware unless some form of emulation or backward compatibility allows the software to run on new hardware.[2]
- These systems can be hard to maintain, improve, and expand because there is a general lack of understanding of the system; the staff who were experts on it have retired or forgotten what they knew about it, and staff who entered the field after it became "legacy" never learned about it in the first place. This can be worsened by lack or loss of documentation. Comair airline company fired its CEO in 2004 due to the failure of an antiquated legacy crew scheduling system that ran into a limitation not known to anyone in the company.[3]
- Legacy systems may have vulnerabilities in older operating systems or applications due to lack of security patches being available or applied. There can also be production configurations that cause security problems. These issues can put the legacy system at risk of being compromised by attackers or knowledgeable insiders.[4]
- Integration with newer systems may also be difficult because new software may use completely different technologies. Integration across technology is quite common in computing, but integration between newer technologies and substantially older ones is not common. There may simply not be sufficient demand for integration technology to be developed. Some of this "glue" code is occasionally developed by vendors and enthusiasts of particular legacy technologies.
- Budgetary constraints often lead corporations to not address the need of replacement or migration of a legacy system. However, companies often don’t consider the increasing supportability costs (people, software and hardware, all mentioned above) and do not take into consideration the enormous loss of capability or business continuity if the legacy system were to fail. Once these considerations are well understood, then based on the proven ROI of a new, more

secure, updated technology stack platform is not as costly as the alternative - and the budget is found.

- Due to the fact that most legacy programmers are entering retirement age and the number of young engineers replacing them is very small, there is an alarming shortage of available workforce. This in turn results in difficulty in maintaining legacy systems, as well as an increase in costs of procuring experienced programmers.[5]

Denne meldingen fra community.spiceworks.com er fra 2016:

«A new report from a congressional group called the Government Accountability Office has found the US government "spends more than 75% of its $80 billion budget ever year on maintaining incredibly outdated technologies and platforms."
Some of these technologies include IBM PCs from the 1970s, which the Department of Defense uses to operate its nuclear arsenal, and the COBOL programming language, which the Social Security systems are based on.
According to Neowin, "The real problems and inefficiencies start piling up when spare parts, updated software or tech support are nearly impossible to find for such old systems." In its report, the Government Accountability Office says the US government is frequently forced to bring people out of retirement to fix antiquated computer systems.»

5.6 Foretakssystemer (ERP)

I løpet av 1990-årene vokste det frem en ny type transaksjonsprosesseringssystemer som integrerte alle de forskjellige funksjonsområdene i ett samlet system. Grunnlaget for denne integreringen er at *forretningsprosessene gjerne går på tvers av funksjonsområdene*. Slike systemer kalles Enterprise Resource Planning systemer, forkortet ERP, eller bare Enterprise Systems. På norsk kan vi kalle dem foretakssystemer (betegnelsen Enterprise Resourse Planning har lite med ressurser og planlegging å gjøre. Betegnelsen har historiske årsaker, nemlig at denne systemtypen hadde sin opprinnelse i MRP-systemer). På slutten av 90-tallet erstattet en rekke store virksomheter sine gamle systemer med foretakssystemer (dette var en forberedelse for årtusenskiftet, da man forventet av mange gamle systemer ville slutte å fungere). Et foretakssystem er en ferdig løsning, men samtidig en løsning som kan tilpasses den enkelte virksomhets behov. Kjente aktører på det internasjonale markedet er SAP, BAAN, PeopleSoft og Oracle. Av disse er SAP størst på ERP (Oracle er større totalt, men hovedproduktet deres er Oracle databaseserver).

Også Microsoft har kommet på banen med foretakssystemer, ved at de har kjøpt opp et annet firma. Deres produkt heter Dynamics AX, og er i følge Microsoft ment for mellomstore og store bedrifter.

Et foretakssystem har en felles, integrert database for alle delsystemer. Hver av funksjonene støttes av en modul, og alle modulene er integrert med databasen og andre moduler. Noen av modulene til SAP/R3 (Jessup 2008):

- Financial accounting (regnskap)
- Human Resources (personaladministrasjon)
- Material Management (materialadministrasjon)
- Production Planning (produksjonsplanlegging)
- Sales & Distribution (salg og distribusjon)

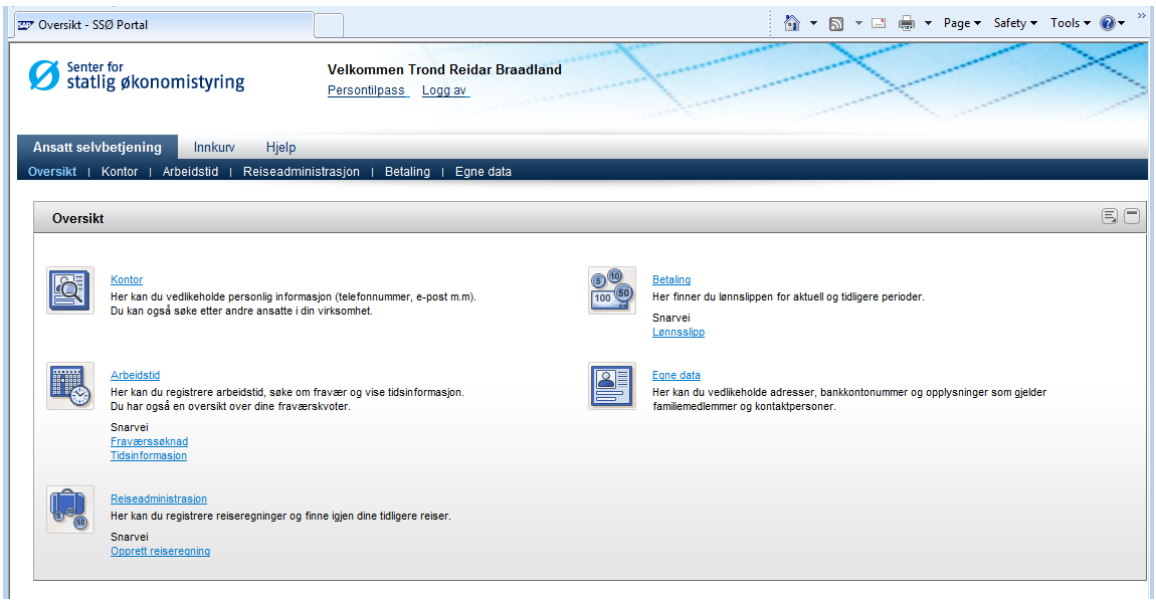
Modulene er ferdig til bruk fra leverandørens side, men de kan også tilpasses den enkelte virksomhets spesielle krav. Dette kan gjøres ved hjelp av ”småmoduler” som legges til modulen, eller ved programmering (SAP kommer med et eget programmeringsspråk kalt ABAP, men dette brukes fortrinnsvis til å lage rapporter). Den vanligste måten å tilpasse på er nok ved avkrysning i tabeller. Leverandørene har også ferdige spesialløsninger tilpasset forskjellige bransjer.

Et eksempel på hvordan et integrert system fungerer, er salg. La oss si at vårt firma får en bestilling fra en kunde, og dette kjøpet skal faktureres. Det som da skjer, er følgende:

- Eksisterende kundes data og kredittverdighet sjekkes
- Det sjekkes om varen er på lager
- Hvis varen er på lager, reserveres den
- ”Plukkliste” lages (dvs en ordre om å hente varen på lager)
- En pakkseddel lages
- Faktura lages
- Faktura legges inn i hovedbok som en utestående fordring
- Ved betaling oppdateres hovedbok
- Hvis kjøpet fører til at varebeholdning synker under bestillingspunkt, genereres en bestilling
- Varesaldo oppdateres med varer i bestilling

En viktig del av innføringen av et foretakssystem er forandring av virksomhetens forretningsprosesser. Et foretakssystem automatiserer svært mye av forretningsprosessene, og de gjør det på en allerede fastlagt måte. I modulene har leverandørene bygget inn måter å gjøre tingene på som er hentet fra studier av ”best practices”, dvs hvordan det gjøres i vellykkede virksomheter. Innføringen av et foretakssystem er derfor også en gjennomgripende forandring av arbeidsmåtene i virksomheten. Disse forandringene er like viktige som det tekniske systemet (ibid.).

Av spesiell interesse for oss er den integrerte databasen. Denne utgjør en felles kilde til informasjon som ledere trenger for å ta beslutninger. Tilgangen til denne databasen er imidlertid ikke enkel. De store systemene, først og fremst SAP/R3 og Oracle, har ekstremt normaliserte databaser. En standardinstallasjon av SAP/R3 har ca. 70000 tabeller! (Casters 2010). I alle fall med SAP/R3 har man heller ikke lov til å hente ut data direkte, men må gå gjennom mellomvare laget for formålet. De standard navnene som brukes på tabeller og felter i disse er heller ikke særlig informative, for eksempel kan en tabell i SAP hete noe slikt som f4211. Som om ikke dette er nok, ligger det meste av metadataene for databasen ikke i databasen selv, men i applikasjonslaget (ibid.). Dette gjør at man er nødt til å gå gjennom SAPs eget grensesnitt for å hente ut data.



Figur 5.2 Statens SAP-baserte system

Figur 5.2 viser det personaladministrative systemet til Senter for statlig økonomistyring, som blant annet brukes av høyskolene. Dette systemet er basert på SAPs modul for personaladministrasjon. Alle lønnsutbetalinger og behandling av reiseregninger håndteres av dette systemet.

Foretakssystemene kan kjøres "rett ut av boksen". Mange bedrifter velger å gjøre nettopp det. Dette er imidlertid ikke nødvendigvis noen optimal løsning. Siden systemene er laget for å kunne brukes i alle slags bedrifter, får man med mye mer enn man egentlig trenger. Systemene kan derfor bli tunge i bruk. Dette kan man løse på tre måter:

- Man kan omprogrammere systemet. SAP har et eget programmeringsspråk som heter ABAP. Dette skaper problemer ved oppgraderinger av systemet. I praksis brukes ABAP fortrinnsvis til å lage rapporter.
- Man kan lage "front-end" løsninger, der man selv programmerer skjermbilder for input, og deretter overfører data til systemet.
- Man krysser av for aktivering eller deaktivering av funksjoner. Dette er den vanligste måten.

Det andre alternativet er tatt i bruk av Kongsberg Automotive, en internasjonal produsent av bildeler med hovedkontor på Kongsberg. De bruker SAP. Men har laget front-end moduler i Excel for strategisk viktige deler av systemet. For eksempel må man i SAP gjennom en rekke forskjellige skjermbilder for å registrere data om en vare, der det for hvert skjermbilde bare er noen få felter som skal fylles ut. Med den egenutviklede front-enden forholder brukerne seg bare til ett enkelt skjermbilde. Dermed spares mye tid, og man har større sikkerhet for at alle registreringer blir korrekte.

Kort kan vi si at fordelene med et ERP er disse:

- Systemet er velprøvet, og man vet hva man får.
- Det finnes en stor brukerbase rundt om i verden, og en rekke diskusjonsfora for disse på internett.
- Leverandøren holder kurs.
- Det er skrevet en rekke bøker om systemene.
- Innebygget best practice kan effektivisere bedriften.
- Leverandøren har erfarne konsulenter som er med i innføringsprosjektet.
- Stadige oppgraderinger av systemet (SAP har regelmessig tre oppgraderinger i året.
- Selv om systemet er standardisert, kan det tilpasses i noen grad.
- Egne bransjemoduler finnes (for eksempel for varehandel, reisebyråer, skipsfart, industri m.m.).
- Kan initiere nødvendige organisasjonsendringer.

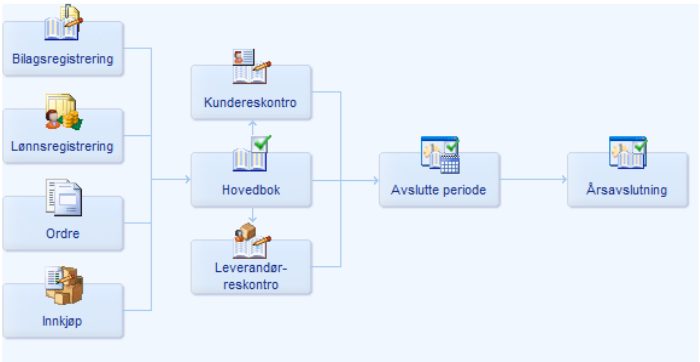
Det er imidlertid også ulemper:

- Systemet er stort og tungt (for eksempel med tusenvis av databasetabeller)
- Man får med mye man ikke trenger.
- Blir ofte dyrere og mer tidkrevende enn antatt.
- Ofte lettere å tilpasse organisasjonen til systemet, noe som ikke alltid er en fordel.
- Sterk avhengighet av leverandør; det er ikke noen liten oppgave å skifte system.
- Brukervennligheten er ofte dårlig (typisk for de store systemene som SAP).

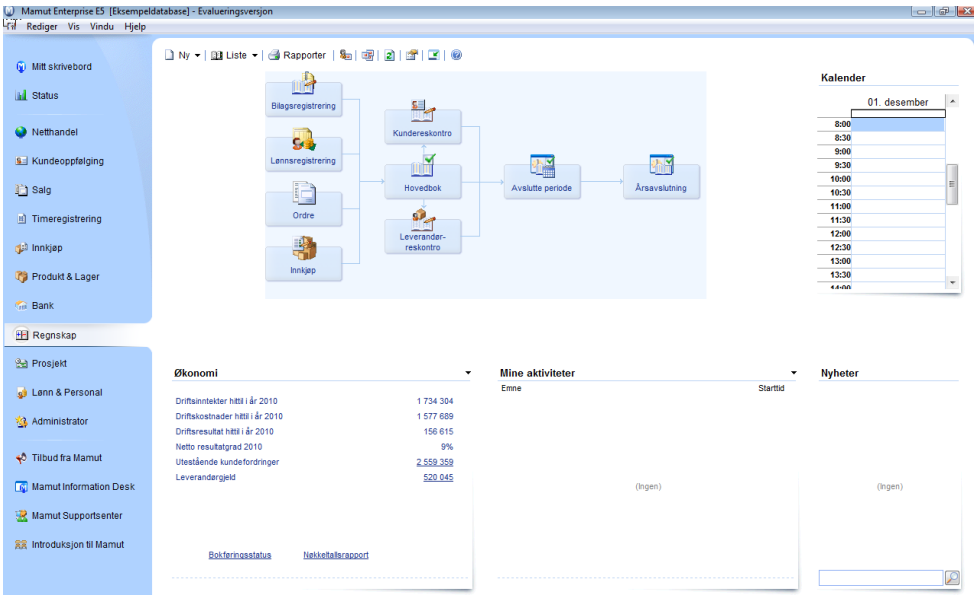
Fordelene er imidlertid så store at et svært stort antall bedrifter har innført slike systemer.

5.7 Eksempel: Mamut Enterprise

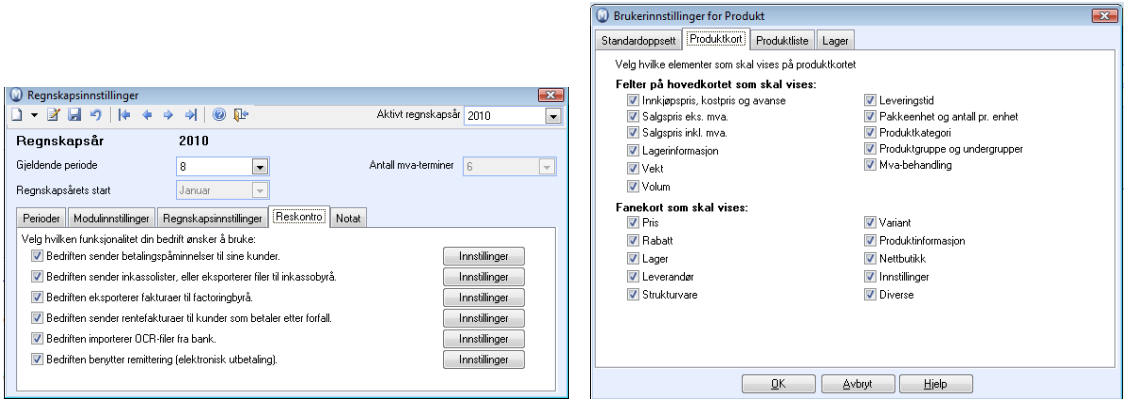
Mamut Enterprise (www.mamut.no) er et norskprodusert ERP-system for små og mellomstore bedrifter. Systemet kan lastes ned i en evalueringsversjon for installasjon på egen maskin (noe som ikke er tilfelle med andre ERP-systemer). En enkel installasjonsprosess og medfølgende "Kom i gang"-dokumentasjon gjør at man raskt kan begynne å teste ut systemet. Mamut utmerker seg også med å ha flytdiagrammer for de enkelte arbeidsområder, noe som gjør det svært oversiktlig å arbeide i.



Figur 5.3 Flytdiagram for et Mamut arbeidsområde



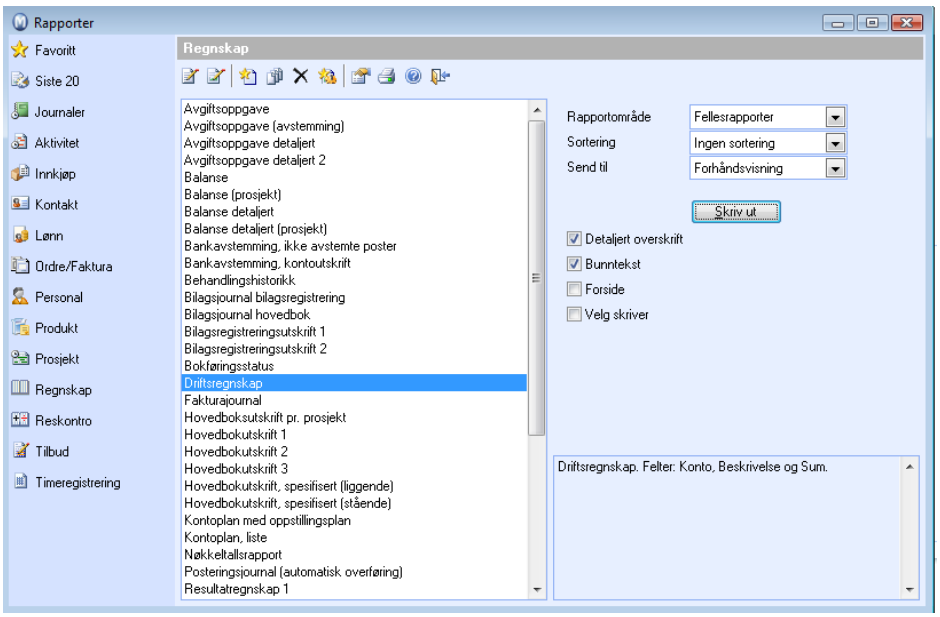
Figur 5.4 Regnskapsmodulen i Mamut



Figur 5.5 Eksempel på innstillinger i Mamut

Når man arbeider i et system som Mamut (eller et av de andre ERP-systemene) kan vi si at "alt henger sammen". Et salg vil for eksempel oppdatere både hovedbok, kundereskontro og lagerbeholdning.

Systemer som dette kommer med et bibliotek av standardrapporter. Disse vil være et utgangspunkt for forretningsanalyse – Business Intelligence. Standardrapportene kan lages med forskjellige utvalg som brukeren bestemmer, og er dermed svært fleksible. Nedenfor ser vi et utvalg av Mamuts standardrapporter for arbeidsområdet **Regnskap**:



Figur 5.6 Rapportbibliotek for regnskap.

Nedenfor er en rapport for driftsregnskap for året 2009 (år velges når man skal skrive ut rapporten) fra dette biblioteket:

Kontormøbler **Driftsregnskap**

Utvalgskriterier: REGNSKAP: Regnskapsår = 2009 (01.01.2009-31.12.2009) og Kilde = 'Hovedbok' (NOK)

Konto	*Beskrivelse*	*Sum*
1510	Kundefordringer	13 766 638,02
1910	Kasse	335 395,46
1930	Bankinnskudd	9 535 460,40
1970	Postgiro	1 689 276,87
2050	Annen egenkapital	-9 852 454,43
2410	Leverandørgjeld	-1 871 431,36
2610	Skattetrekk	-4 005 325,00
2710	Utgående, høy mva	
2720	Inngående, høy mva	
2750	Oppgjørskonto merverdiavgift	-4 704 728,00
2780	Skyldig arbeidsgiveravgift	-981 783,00
2781	Arb.giv.avg. pål. feriep.	-117 813,96
2910	Skyldig lønn	-2 957 675,00
2920	Skyldig feriepenger	-835 560,00
	Sum:	0,00
3010	Salgsinntekter, høy mva	-9 526 473,43
3110	Salgsinntekter, avgiftsfrie	-3 346,00
4010	Innkjøp varer, avgiftspliktig, høy mva	1 500 852,15
4110	Innkjøp varer, avgiftsfritt	76 535,76
5010	Faste lønninger	3 522 000,00
5190	Påløpne feriepenger	422 640,00
5410	Arbeidsgiveravgift	496 602,00
5411	Arb.giv.avg. pål. feriep.	59 592,24
7105	Øreavrunding	-32,90
7770	Bank og kortgebyrer	392,00
8080	Agio gevinst	-392,00
8980	Avsatt til fri egenkapital	3 451 630,18
	Sum:	0,00

Figur 5.7 Rapport fra Mamut

5.8 Programvare for kunderelasjonshåndtering

Kunderelasjonshåndtering, på engelsk Customer Relationship Management, er et konsept innen faget markedsføring. Essensen i kunderelasjonshåndtering er å skape langsiktige kunder, ut fra erkjennelsen at det er mer kostnadseffektivt å beholde kunder enn å skaffe nye. Kundene må derfor pleies og følges opp best mulig. Ideen er å gi kundene det de vil ha i en slik grad at de ikke vil henvende seg andre steder.

Kunderelasjonshåndtering har i utgangspunktet ikke noe med IT å gjøre, men brukes i dag ofte synonymt med en gruppe programvare. For å gjennomføre kunderelasjonshåndtering i praksis trenger virksomheten noe som gjør det mulig å håndtere all informasjon om kundene på ett sted.

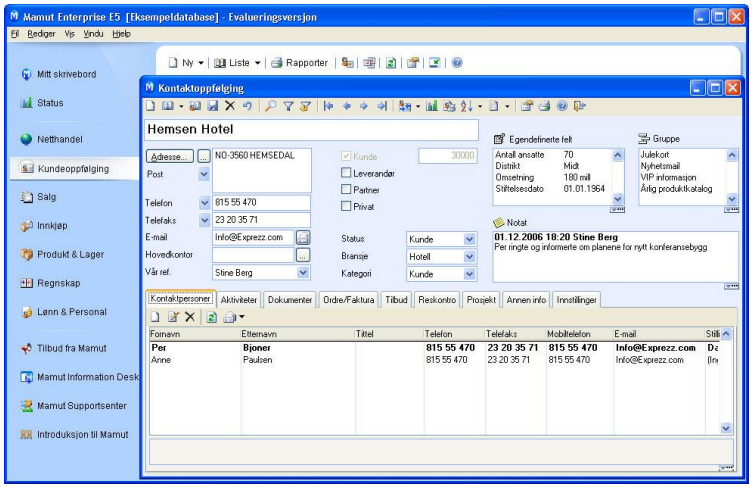
Det finnes et stort antall programmer for kunderelasjonshåndtering på markedet, fra enkle programmer beregnet på små bedrifter til omfattende systemer beregnet på de store virksomhetene. Disse er gjerne integrert med virksomhetens ERP-system.

Et komplett kunderelasjonshåndteringssystem består av en operasjonell del og en analytisk del (Laudon & Laudon). Andre forfattere legger til enten en strategisk del eller en samarbeidsdel (Jessup 2008). Vi skal her konsentrere oss om de to første.

Den operasjonelle delen av et system for kunderelasjonshåndtering har en rekke funksjoner for daglig bruk mot kundene, som disse (Laudon & Laudon):

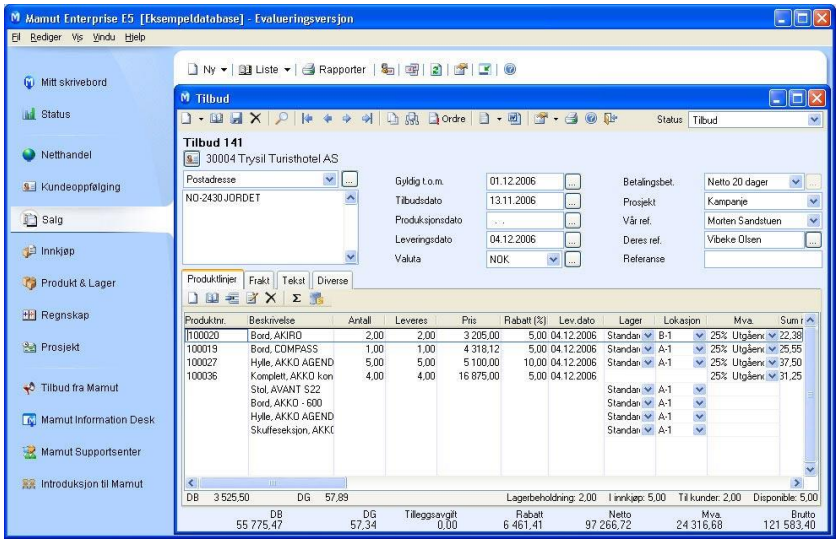
- Håndtere tilbud
- Behandle og følge ordre
- Kontakt med kunder
- Oppfølging av prosjekter
- Følge kundens tidligere kjøpshistorie
- Kundestøtte
- Håndtering av kontrakter

Et skjermbilde som brukes i oppfølging av kundekontakter kan se slik ut (Mamut Enterprise):



Figur 5.8 Skjermbilde for kundekontakter (Mamut Enterprise)

Her ligger alle data om kunden, kontaktpersoner og hva som har skjedd, samlet på ett sted. Virksomheten er dermed ikke avhengig av den enkelte selgers personlige notater.



Figur 5.9 Skjermbilde for tilbud til kunde (Mamut Enterprise)

Den delen av et CRM-system som brukes i det daglige salgsarbeidet kalles også for et **front office** system. Dette i motsetning til den analytiske delen, som betegnes som et **back office** system. Den analytiske delen brukes ikke i den daglige kontakten med kunden. I stedet kjøres her analyser der man fokuserer på forhold som disse (Jessup 2008):

- Markedskampanjer
- Kundesegmentering
- Prisanalyser
- Risikohåndtering
- Konkurrentanalyser
- Kundetilfredshet
- Produktenes livssyklus

Den analytiske delen av systemet utfører analyser som omfattes av virksomhetens Business Intelligence-system. Samtidig vil mye av informasjonen i CRM-systemet ligge i de operative systemene, i praksis gjerne et ERP-system. Et system for kunderelasjonshåndtering kan dermed ses som en bro mellom ERP-systemet og BI-systemet.

Noen leverandører av CRM programvare:

www.superoffice.no er leverandøren av SuperOffice, kanskje den mest populære frittstående applikasjonen her i landet.

www.visma.no Et ERP-system som nå bruker SuperOffice som CRM-modul, men som også har en egen CRM-modul.

www.mamut.com Et ERP-system med egen CRM-modul.

www.sap.no SAP er markedsleder blant de store bedriftene. Den har sin egen CRM-modul.

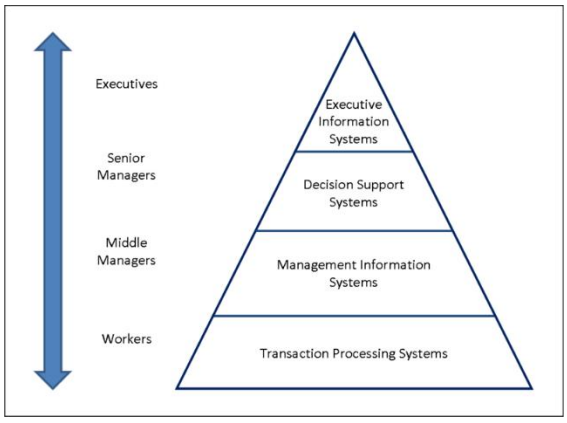
Kapittel 6 Historiske begreper: Systemer for ledere

Transaksjonsprosesseringssystemene som ble presentert i kapittel 4 er ikke spesielt velegnet til å forsyne ledelsen med informasjon. Det er mange årsaker til dette. En årsak er at disse systemene er kritiske for den daglige driften. Man kan ikke risikere avbrudd eller forsinkelser i behandlingen av transaksjoner på grunn av rapportering. En annen årsak er at transaksjonsprosesseringssystemene er optimalisert nettopp for transaksjonsprosessering. Det innebærer at de har en normalisert database, der rapportering vil kreve mye og kompliserte join. I tillegg til dette inneholder disse systemene i begrenset grad historiske data, og det er ofte slike data som er av interesse for ledelsen.

For å bøte på de manglene transaksjonsprosesseringssystemene har når det gjelder rapportering og analyse, er det laget andre systemtyper. Felles for disse er at de henter mye av sine data fra transaksjonsprosesseringssystemene, men siden de ellers er uavhengige av disse, kan de optimaliseres for spørringer og analyser.

I dette kapittelet beskrives det som vi kan kalle *idealtyper* av systemer: Management Information Systems, beslutningsstøttesystemer og Executive Information Systems. Et virkelig system, slik vi finner dem implementert i virksomhetene, kan godt kombinere egenskaper fra disse typene. Det er imidlertid gode grunner til å gi detaljerte beskrivelser av idealtypene. Dels stiller de forskjellig krav til verktøyene som brukes for å lage dem, dels brukes de til forskjellige typer beslutninger.

Vi kan bruke organisasjonspyramiden fra kapittel 3 til å vise systemtyper på de forskjellige nivåene. I bunnen ligger transaksjonsprosesseringssystemene, som er systemene som støtter den daglige driften. Deretter har vi de tre typene informasjonssystemer vi skal se på i dette kapittelet (Laudon 2017):



Figur 6.1 Typer informasjonssystem på forskjellige organisasjonsnivåer (Illustrasjon: Compo, Wikimedia Commons https://commons.wikimedia.org/wiki/File:Four-Level-Pyramid-model.png)

6.1 Management Information Systems

Management Information Systems (MIS) er systemer beregnet på mellomlederes behov for kontroll av virksomheten. Dette er altså systemer på taktisk nivå (mellomledernivå), og vi kan også kalle dem taktiske informasjonssystemer (Gottschalk 2002).

Mellomledersystemene er først og fremst beregnet på å støtte strukturerte beslutningsprosesser (kontroll av aktiviteter kan ses som strukturerte beslutningsprosesser), det vil si at reglene for beslutningene er gitt. Disse systemene er typisk basert på ferdige rapporter. Siden regler og informasjonsbehov er kjent, kan man lage ferdige rapporter. Når disse skal kjøres, trenger brukeren bare oppgi parametere som periode, eventuelt salgsdistrikt, utvalg produkter, utvalg kunder osv.

Bruken av ferdige rapporter gjør at et Management Information System ikke behøver ha sin egen database. De standardiserte rapportene er basert på programkode, og man kjenner nøyaktig hvor mye maskinressurser hver rapport vil bruke. I et flerbrukermiljø kan rapportene kjøres som batch-jobber, det vil si at operativsystemet tildeler prosessortid når det er ledig kapasitet på maskinen. Dette krever imidlertid at brukerne har svært begrensede muligheter til å definere egne rapporter.

6.2 Beslutningsstøttesystemer

Når man ikke kan forutsi hva slags informasjon man har bruk for, trenger man helt andre typer systemer enn de som er beskrevet foran. Et beslutningsstøttesystem – decision support system (DSS) på engelsk – skal støtte semistrukturerte beslutningsprosesser med informasjon. Typisk vil beslutningsstøttesystemet støtte de strukturerte delene av prosessen. På den måten får man også isolert de delene av beslutningen som er basert på erfaring og skjønn (Alter 2002).

Organisatorisk dekker beslutningsstøttesystemene deler av mellomledernivået og deler av toppledernivået.

Tradisjonelt skiller man mellom to hovedtyper beslutningsstøttesystemer: **modelldrevne** og **datadrevne** (Laudon 2002). I dag kan vi også ta med en tredje type: "**kunstig intelligens"-drevne** (Alter 2002).

Et **modelldrevet** beslutningsstøttesystem er basert på bruk av matematiske modeller. Disse modellene beskriver kjent eller antatt oppførsel for et system. Modellene implementeres enten med et regneark eller et modellspråk. Ofte er disse såkalt stand-alone, dvs de er ikke tilknyttet en database.

Et **datadrevet** beslutningsstøttesystem er basert på analyse av store datamengder. Disse systemene bruker typisk en egen database isolert fra transaksjonsprosesseringssystemene. Analysene kan stille store krav til kapasitet hos maskinvare og programvare.

Kunstig intelligens-drevne beslutningsstøttesystemer er basert på teknikker innen det som kalles kunstig intelligens-forskning (AI – Artificial Intelligence). Det er flere svært forskjellige teknikker det her er snakk om, men alle kombinerer data med bruk av logikk i en eller annen form.

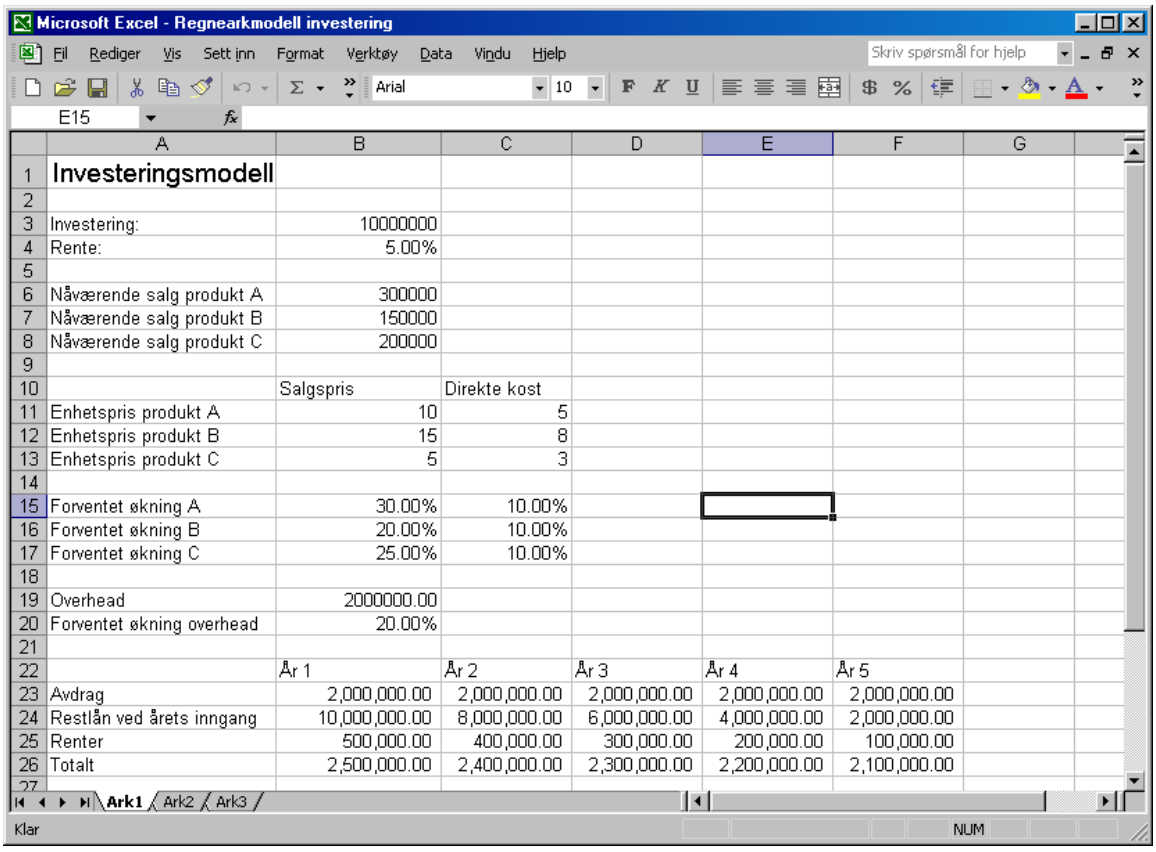
Innenfor disse tre typene beslutningsstøttesystemer brukes flere forskjellige analysemetoder (Alter 2002), som vist i tabell 7.1.

En arkitektur for et beslutningsstøttesystem består av flere elementer:

- En database med data som skal analyseres. Dette kan være en spesielt strukturert database som et datavarehus, eller rett og slett data som tastes inn i et regneark
- En modellbase med modeller som kan brukes
- Programvare for analyse, som kan være basert på formler eller logiske slutninger
- Eventuelt en regelbase for ekspertsystemer og fuzzy logikk
- Brukergrensesnitt for interaktiv bruk

6.2.1 Simulering og optimering

Både ved simulering og optimering brukes modellverktøy, det vil si språk som kan brukes til å lage en modell av beslutningsproblemet. Mest kjent av slike verktøy er regnearket, som kan brukes både til simuleringer og optimering. Til mer avanserte modeller finnes modellspråk, som for eksempel PowerSim. Disse har en høyere brukerterskel, men er mer avanserte enn regnearkene. Spesielt gjelder dette på optimeringssituasjoner.



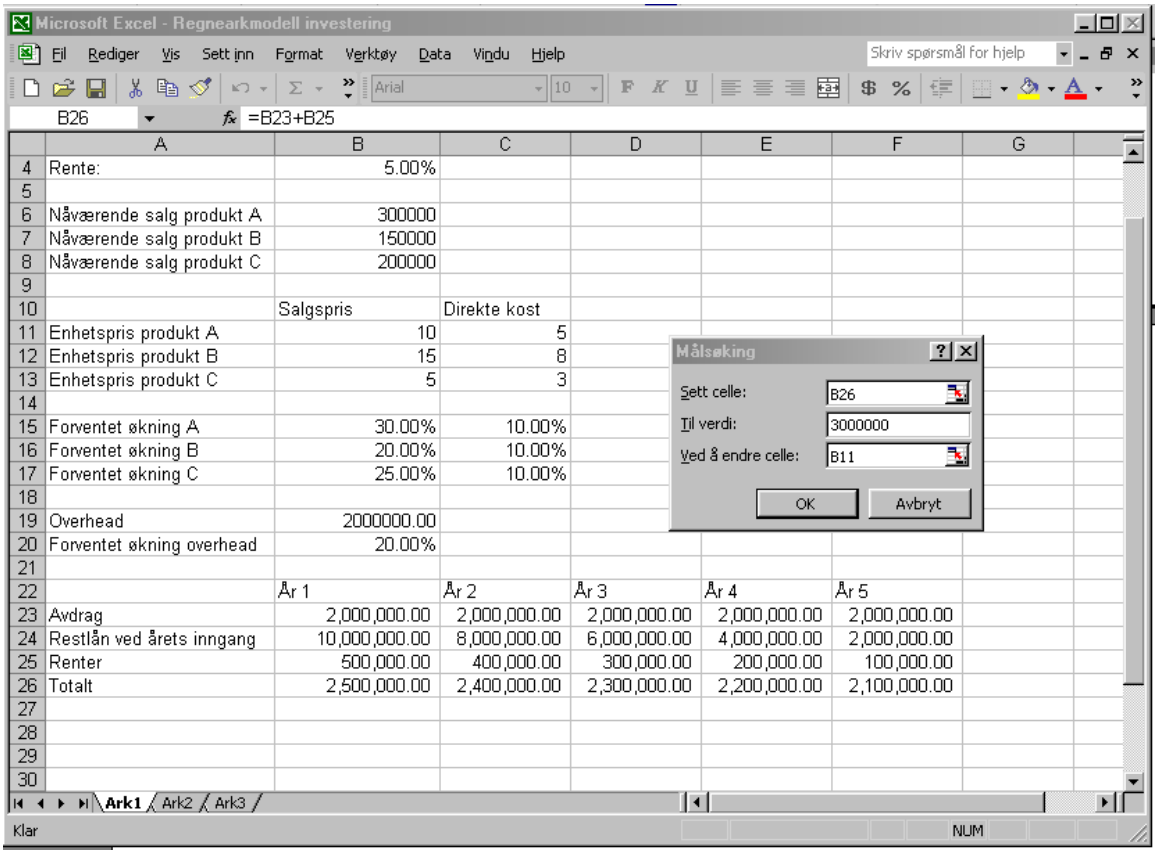
	A	B	C	D	E	F	G
1	**Investeringsmodell**						
2							
3	Investering:	10000000					
4	Rente:	5.00%					
5							
6	Nåværende salg produkt A	300000					
7	Nåværende salg produkt B	150000					
8	Nåværende salg produkt C	200000					
9							
10		Salgspris	Direkte kost				
11	Enhetspris produkt A	10	5				
12	Enhetspris produkt B	15	8				
13	Enhetspris produkt C	5	3				
14							
15	Forventet økning A	30.00%	10.00%				
16	Forventet økning B	20.00%	10.00%				
17	Forventet økning C	25.00%	10.00%				
18							
19	Overhead	2000000.00					
20	Forventet økning overhead	20.00%					
21							
22		År 1	År 2	År 3	År 4	År 5	
23	Avdrag	2,000,000.00	2,000,000.00	2,000,000.00	2,000,000.00	2,000,000.00	
24	Restlån ved årets inngang	10,000,000.00	8,000,000.00	6,000,000.00	4,000,000.00	2,000,000.00	
25	Renter	500,000.00	400,000.00	300,000.00	200,000.00	100,000.00	
26	Totalt	2,500,000.00	2,400,000.00	2,300,000.00	2,200,000.00	2,100,000.00	

Figur 6.2 Simulering med regneark

Ved simulering lager man en modell med forskjellige variabler som antall solgt, salgspris pr enhet, innkjøpspris, rente osv. Modellen presenterer så resultater i form av for eksempel fortjeneste. En enkel simuleringsmodell er vist i figur 6.2.

Ved optimering arbeider man motsatt av hva man gjør ved simulering. Nå oppgir man hva man ønsker som resultat, og modellverktøyet vil så beregne optimale verdier for forskjellige variabler. For at dette skal være mulig, må det oppgis betingelser for de

enkelte variabler. Regneark som Excel har mulighet for slike analyser, men det er større muligheter med modellspråk som PowerSim og GPSS. En måte å gjøre dette på i Excel, er ved målsøking (Verktøy | Målsøking).



	A	B	C	D	E	F	G
4	Rente:	5.00%					
5							
6	Nåværende salg produkt A	300000					
7	Nåværende salg produkt B	150000					
8	Nåværende salg produkt C	200000					
9							
10		Salgspris	Direkte kost				
11	Enhetspris produkt A	10	5				
12	Enhetspris produkt B	15	8				
13	Enhetspris produkt C	5	3				
14							
15	Forventet økning A	30.00%	10.00%				
16	Forventet økning B	20.00%	10.00%				
17	Forventet økning C	25.00%	10.00%				
18							
19	Overhead	2000000.00					
20	Forventet økning overhead	20.00%					
21							
22		År 1	År 2	År 3	År 4	År 5	
23	Avdrag	2,000,000.00	2,000,000.00	2,000,000.00	2,000,000.00	2,000,000.00	
24	Restlån ved årets inngang	10,000,000.00	8,000,000.00	6,000,000.00	4,000,000.00	2,000,000.00	
25	Renter	500,000.00	400,000.00	300,000.00	200,000.00	100,000.00	
26	Totalt	2,500,000.00	2,400,000.00	2,300,000.00	2,200,000.00	2,100,000.00	
27							
28							
29							
30							

Figur 6.3 Målsøking i Excel

6.2.2 Data Mining og OLAP

Online Analytical Processing (OLAP) er analyse av store mengder transaksjonsdata ved hjelp av egne verktøy. OLAP omfatter utvalg, summeringer og drilling. Vi kommer nærmere inn på dette i kapitlene om datavarehus. Transaksjonsdataene som er grunnlaget for OLAP befinner seg typisk i et datavarehus.

Data Mining er bruk av statistiske metoder for å finne sammenhenger i store mengder med transaksjonsdata. Hensikten med slike analyser er vanligvis å skaffe informasjon som kan brukes i markedsføring.

6.2.3 Kunnskapsteknologi

Ekspertsystemer, nevrale nett, fuzzy logikk, eksempelbasert resonnering og intelligente agenter er alle eksempler på såkalt kunnskapsteknologi, også kalt ”kunstig intelligens”. Vi skal komme nærmere inn på denne teknologien i et senere kapittel. Foreløpig skal vi nøye oss med å nevne at bruk av ”kunstig intelligens” etter hvert er blitt vanlig i sluttbrukerapplikasjoner beregnet på ledere og analytikere. Spesielt innenfor Data Mining brukes denne teknologien. Man kan også ha rådgivende systemer, såkalte ekspertsystemer.

6.3 Executive Information Systems

Begrepet Executive Information System oppstod på 1980-tallet. Definisjonen på et slikt system var ”a computerized system that provides executives with easy access to internal and external information that is relevant to their critical success factors” (Watson 1997:3). Denne definisjonen er litt for overordnet til at vi får noe inntrykk av hva et slikt system er. Hvis vi derimot ser på egenskapene et EIS må ha, får vi et klarere inntrykk (ibid.):

- Et EIS kan skreddersys den enkelte bruker
- mulighet for både numeriske data (kalt ”harde data”) og tekstlig informasjon (kalt ”myke data”)
- Systemet kan ekstrahere, filtrere, komprimere og spore kritiske data
- Kan gi online tilgang til status, trender, avvik
- Gir mulighet til drilling
- Kan gi tilgang til og integrere et bredt spektrum av interne og eksterne data
- Er brukervennlig og krever lite opplæring
- Kan brukes direkte av toppledere uten mellommenn
- Presenterer informasjon i form av grafikk, tabeller og tekst

Ofte brukes betegnelsen Executive Support System – ESS – i stedet for EIS. Watson et al. Argumenterer for at ESS er et noe bredere begrep enn EIS, idet et ESS inneholder en eller flere av følgende egenskaper:

- elektronisk kommunikasjon som e-mail og gruppevare
- analysemuligheter som regneark, modellverktøy og spørrespråk
- organiseringsverktøy som elektronisk kalender og personlig arkiv

Dette skillet er antagelig noe oppkonstruert, idet de fleste moderne applikasjoner beregnet for ledere (og for den slags skyld de fleste andre brukere) vil inneholde disse tilleggsfunksjonene.

Det sentrale som skiller EIS/ESS fra ordinære beslutningsstøttesystemer er brukervennligheten. Et beslutningsstøttesystem er beregnet brukt av analytikere, og brukergrensesnittet behøver ikke være spesielt brukervennlig. Et EIS skal brukes av folk som ikke har tid eller motivasjon til å lære seg et komplisert brukergrensesnitt. Toppledere har dårlig tid, og vil ha det så enkelt som mulig (og dette er ikke ironisk ment).

Opprinnelsen til EIS var i en tid da stormaskiner og applikasjoner for disse var enerådende. Et MIS var typisk basert på COBOL-rapporter, som beskrevet tidligere. Topplederne krevde tilgang til verktøy som kunne gi dem informasjon i sanntid (online). Dette var krevende med datidens teknologi, og kostnadene ved å utvikle og implementere slike systemer var store. De ble dermed også forbeholdt toppledelsen.

I dag har kanskje ikke begrepet EIS så mye for seg. Moderne utviklingsverktøy gjør det relativt enkelt å utvikle og tilpasse individuelt applikasjoner av typen EIS. En av de store

produsentene av slike verktøy, SAS Institute, annonserte for noen år siden at for deres EIS utviklingsverktøy stod EIS ikke lenger for ”Executive Information System”, men for ”Everybody’s Information System”.

Kapittel 7 Balansert målstyring

De tradisjonelle styringsverktøyene for bedrifter har vært rene økonomiske. De senere årene er det blitt utviklet flere forskjellige styringsverktøy som bedre skal passe moderne bedrifter. Det mest utbredte av disse er balansert målstyring.

7.1 Bakgrunnen for balansert målstyring

Balansert målstyring – på engelsk Balanced Scorecard - ble introdusert av Robert S. Kaplan og David P. Norton på 1990-tallet. Utgangspunktet deres er at dagens bedrifter opererer i et informasjonssamfunn, i motsetning til tidligere tiders industrisamfunn (de regner at denne overgangen skjedde på midten av 1970-tallet). I industrisamfunnet konkurrerte bedrifter hovedsakelig ved masseproduksjon, dvs en kostnadslederstrategi. Styringssystemene som ble brukt var først og fremst økonomiske målinger. Konkurransen var heller ikke så stor, siden mange bedridter opererte i nasjonale markeder. Innen mange sektorer, som kommunikasjon, transport og finans, var det sterke reguleringer som sikret mer eller mindre monopol (som det gamle Televerket i Norge, som hadde monopol på teletjenester).

Kaplan og Norton har satt opp en liste over forutsetninger som nå er helt endret (Kaplan 1996):

Integrerte forretningsprosesser: Tidligere var bedriftene opptatt av å optimalisere funksjonene (produksjon, salg, distribusjon osv). I informasjonsalderen er fokus i stedet på integrerte forretningsprosesser, som gjerne går på tvers av funksjonsområdene.
Kobling til kunder og leverandører: Tidligere produserte bedriftene etter prognoser og produksjonsplaner. med dagens teknologi kan bedriftene i stededet la kundenes ordre utløse både innkjøp, produksjon og distribusjon. De får dermed et integrert system fra leverandører gjennom hele verdikjeden til kunde.
Kundesegmentering: I masseproduksjonens barndom var fokuset på standardiserte produkter. henry Fords T-Ford var et ekstremt eksempel - den kunne fås i alle farger så lenge den var sort. I dag kan bedriftene rette seg mot en rekke forskjellige kundesegmenter, og den overordnede strategien er i stor grad differensiering.
Global konkurranse: Tollbarrierer og subsidier gjorde at de fleste bedrifter opererte i begrensede markeder. I dag er de fleste slike ordninger borte (vi kan bare nevne stikkord som WTO, EU, EØS), og konkurransen er i stor grad global. Norske bedrifter som Kongsberg Automotive og Raufoss Technology leverer for eksempel deler til bilindustrien. Bedrifter som Yara og Norske Skog har fabrikker i mange land.
Innovasjon: Produktenes livssyklus er mye kortere i dag. Dette gjør at bedriftene må være gode på både å forutse kundenes behov og utvikle nye produkter raskt, for deretter å få dem raskt i produksjon.
Kunnskapsarbeid: I industrisamfunnet var det et skarpt skille mellom ledelse og ingeniører på den ene siden og vanlige arbeidere på den andre. I dagens bedrifter er antallet "rene" arbeidere mye mindre, mens det er en stor andel personer innen kunnskapsarbeid, som produktutvikling, markedsføring, kundeservice og administrasjon. De ansattes kunnskaper er av stor betydning, og dette er kunnskap som både må ivaretas og utvikles videre.

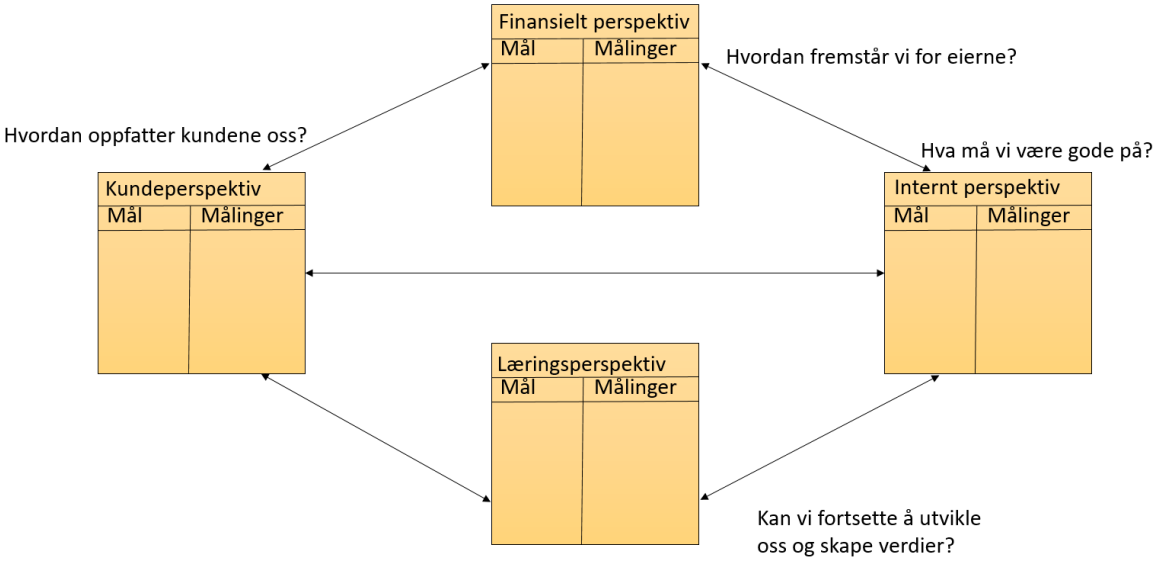
I det hele tatt er verden mye mer kompleks for informasjonsalderens bedrifter. Det gjør også at industrialderens ledelsesverktøy ikke lenger er tilstrekkelig. Balansert målstyring er et ledelsesverktøy som skal oppfylle vår tids behov. Enkelt sagt kombinerer balansert målstyring finansielle målinger, som er målinger av hva som har skjedd, med målinger av driverne for fremtidig ytelse.

7.2 Hva er balansert målstyring?

Balansert målstyring er et system for strategisk ledelse. Utgangspunktet er bedriftens visjon og strategi. Disse må så operasjonaliseres. Dette gjøres ved å anlegge fire forskjellige perspektiver:

- Kundeperspektiv
- Finansielt perspektiv
- Prosessperspektiv
- Læring og vekst-perspektiv

Kaplan og Norton fremstiller dette i følgende figur:



Figur 7.1 Balansert målstyring (etter www.balancedscorecard.org)

Vi starter altså med bedriftens visjon. Denne uttrykker hva vi ønsker at bedriften skal være. Eksempler på visjoner er. "Vi skal være den største sjokoladeprodusenten i Nord-Europa", "Vi skal være den største sportskjeden i Norge", "Vi skal være det foretrukne valg blandt landets konsulentselskaper". Deretter utarbeides en strategi for hvordan denne visjonen skal nås.

For mange bedrifter har visjonen lett for å bli på festtalenivå. Utfordringen er å "oversette" visjonen til konkrete handlinger for den enkelte i bedriften. Det er dette strategiarbeidet skal gjøre, men ofte blir ikke strategien konkret nok. Balansert målstyring er en metode for å få til dette.

Figur 8.1 viser de fire perspektivene i balansert målstyring og hvordan de henger sammen med hverandre og med visjon og strategi. For hvert perspektiv skal man konkretisere mål

(objectives), målinger (measures), målområder (targets) og initiativer. Vi skal se nærmere på dette.

7.2.1 Kundeperspektiv

I kundeperspektivet identifiseres kundegrupper og markeder og hvordan man kan måle ytelsen i disse. Typiske målinger her er kundetilfredshet, kundelojalitet, evne til å tiltrekke nye kunder, lønnsomhet for de enkelte kundegrupper og markedsandeler inne nde forskjellige segmentene. Målinger vil også omfatte mer spesifikke ting som bidrar til tilfredshet og lojalitet, som leveringstid, evne til å lansere nye produkter og mye annet.

Måten man kan måle dette på er mange. Kundetilfredshet og lojalitet kan måles gjennom spørreundersøkelser, men også antall gjenkjøp vil gi en god indikasjon. Vekst i eller tap av markedsandeler i forhold til konkurrentene likeså. Den enkelte bedrift må utarbeide en liste ut fra sin bransje og valgte markeder.

7.2.2 Finansielt perspektiv

Det finansielle perspektivet er både kortsiktig og langsiktig. Også i dag er det mange bedrifter som arbeider utelukkende ut fra et kortsiktig perspektiv, ofte knyttet til lederes bonusordninger. ved å foreta disposisjoner som gir en kortsiktig økning av aksjekursene, kan lederne innkassere fete bonuser. Særlig amerikanske bedrifter gir mange ferske eksempler på at dette kan gå virkelig galt. Tiltak som gir kortsiktige gevinster kan ofte være ødeleggende på lang sikt. Et eksempel på dette er reduksjon i arbeidsstyrken, som sparer penger på kort sikt, men som også gjør at bedriften kan miste viktig kompetanse. Et annet eksempel kan være en bedrift som er så dominerende i markedet at den utnytter kundene. Den dagen kundene har et alternativ forsvinner disse. Mye av problemene SAS har kan forklares med dette. I balansert målstyring er det derfor et prinsipp at man også måler det som sikrer bedriften vekst over tid.

Det finansielle perspektivet skal måle hvordan forskjellige tiltak bidrar til bedriftens resultat. Typiske målinger er avkastning på kapital, omsetning og kostnader. Andre målinger kan være vekst i omsetning og verdiskaping.

7.2.3 Interne prosesser perspektiv

Det er de interne prosessene som skal sette bedriften i stand til å konkurrere. Det er derfor viktig å måle effektiviteten av disse, for å finne ut hvor forbedringer kan settes inn. Et viktig prinsipp i balansert målstyring er at man ikke bare skal måle effektiviteten av eksisterende prosesser, men også komme opp med helt nye prosesser for å oppfylle bedriftens strategi. Et annet viktig prinsipp er å inkludere innovasjonsprosesser blant forretningsprosessene. Dette har igjen sammenheng med moderne bedrifters behov for kontinuerlig innovasjon. Her kan for eksempel bedriftens evne til styring av utviklingsprosjekter være en kritisk faktor.

7.2.4 Læring og vekst perspektivet

For å sikre langsiktig suksess må bedriften legge vekt på organisasjonslæring. Det er tre kilder til læring og vekst:

- Mennesker
- Systemer
- Prosedyrer

Kompetansen til menneskene i bedriften blir stadig viktigere, og dette har konsekvenser både for rekruttering og å ta vare på ansatte man allerede har. Disse skal både trives og få anledning til å utvikle seg. Systemer vil her i stor grad dreie seg om informasjonssystemer. Prosedyrer er detaljerte regler for hvordan man skal utføre arbeidet.

Målinger som kan gjøres på dette området omfatter hvor fornøyde de ansatte er, hvilken kompetanse de har, turnover (hvor ofte ansatte slutter), eller hvor lett tilgjengelig informasjon er i beslutningssituasjoner.

I balansert målstyring skal man utarbeide *målekort* (det er dette som kalles scorecards på engelsk). Målekortene skal inneholde målsetninger og hvilke parametere man skal bruke for å måle disse. Videre må man ha normalområder og varselområder for avleste verdier, og også viktigheten av de enkelte måleparametere.

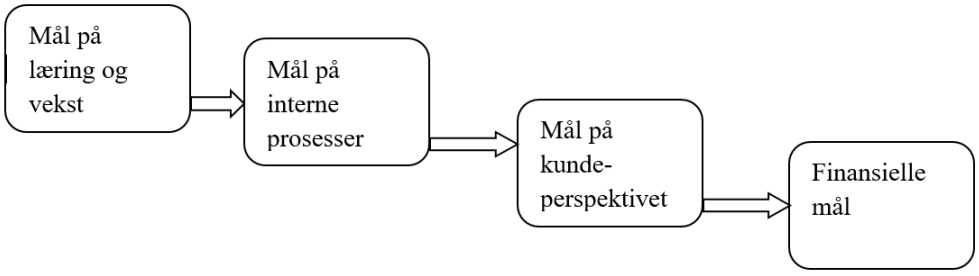
At målekortene skal være balanserte, betyr at det skal være balanse mellom:

- Kortsiktige og langsiktige perspektiver.
- Eksterne mål for eiere og kunder og interne mål for forretningsprosesser og læring og vekst.
- Ønsket utfall og driverne for de utfallene.

Det er ingen god idé å sette i gang med å lage målekort for hele bedriften med en gang. Kaplan og Norton anbefaler å starte med et forretningsområde, eller strategisk forretningsenhet (Strategic Business Unit – BSC).

Kaplan og Norton legger stor vekt på årsakssammenhenger i bedriftens ytelse (Kaplan 1996). Slike årsakssammenhenger skal gjenspeiles i målekortene, dvs det må fremgå hvordan et resultat og driverne av dette resultatet henger sammen.

Kaplan og Norton regner selv med følgende årsakssammenheng mellom målene fra de fire perspektivene (ibid.):



Figur 7.2 Årsakssammenhenger mellom perspektivene

Ut fra målekortene tegner man nå strategiske kart som viser sammenhenger mellom elementer innen de forskjellige perspektivene.

Bedriften må nå fylle ut dette med sine egne elementer. La oss lage et eksempel.

La oss si at vi har en bedrift som produserer sengetøy i Norge og leverer dem til norske butikkjeder. Konkurrentene har stort sett plassert produksjonen i lavkostland i Asia. Problemet for disse er at de må produsere store serier som på grunn av transport bruker måneder på å komme på markedet i Norge. Man må dermed kjøpe inn store partier uten å vite hvordan de vil slå an i markedet. Vår bedrift satser i stedet på å levere produkter som til enhver tid er tilpasset markedets etterspørsel.

De finansielle målene er å øke fortjenesten og dermed også aksjekursen. For å få til dette trenger man å ta kunder fra konkurrentene. Et annet virkemiddel er å få hver kunde til å kjøpe mer. Dette igjen krever at man raskt kan produsere nye partier, som ikke engang trenger være så store (man kan jo teste ut markedet først). Produserte varer må også raskt komme ut på markedet, noe som krever god logistikk. Tilpasset distribusjon betyr også at man må ha mulighet til raskt å designe nye mønstre osv. Dette igjen krever at man har kompetanse både på markedsanalyser og design. Ut fra denne beskrivelsen kan vå nå lage følgende strategiske kart:

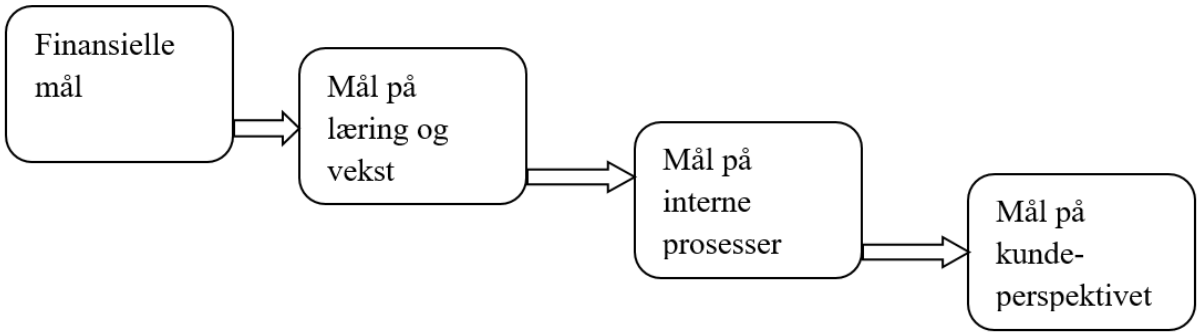
Det strategiske kartet i seg selv er ikke alt. I tillegg må man nå sette opp en oversikt over målsettinger, hvordan man kan måle disse, hvilke verdier vi ønsker oss samt tiltak for å oppnå dette. En slikt oppsett er vist i tabell 7.1.

Perspektiv	Mål	Måling	Målsetning	Tiltak
Finansielt	Øke profitt Øke aksjekurs	Return on investment Kurs på Oslo Børs	Opp 10 % Opp 5 %	
Kunde	Få større markedsandel Få kundene til å kjøpe mer	% markedsandel Salg per kunde	Opp 10 % Opp 20 %	Presentere vårt konsept for butikker Lage konsepter
Interne prosesser	Effektivisere distribusjon Tilpasse produksjon til nye serier Designe nye produkter	Tid fra fabrikk til butikk Tid for omlegging Tid for ny design	Maks 2 døgn Maks 1 døgn Maks 1 uke	Samarbeid med transportør Maskiner styrt av DAK-system DAK-system
Læring	Mer kunnskap om kunder og markeder Ypperlige kunnskaper om design	Tid for å analysere markeder Evne til å lage ny design	Maks 1 dag Utdannelse Antall opp 25 % Utdannelse	BI-system Ansette analytiker Kursing Ansette designer Kursing

Tabell 7.1 Tabell avledet av strategisk kart

Analysen kan her trekkes enda lengre, med for eksempel vekting av forskjellige elementer, varselverdier m.m. Vi skal ikke gå videre på dette her.

Det hierarkiet vi har satt opp for de fire perspektivene er egentlig beregnet for bedrifter, der inntjening alltid vil være et endelig mål- For virksomheter som universiteter og høyskoler kan vi sette opp en annen årsakssammenheng:



Figur 7.3 Årsakssammenhenger for en høyskole

Dette er fordi en høyskole ikke tjener penger, men får bevilgninger over statsbudsjettet.

7.3 Arbeidsprosess

Det finnes mange "oppskrifter" på hvordan man bør arbeide med målekortene. På nettsidene til The Balanced Scorecard Institute (www.balancedscorecard.org) anbefales en kontinuerlig prosess med ni trinn - Nine Steps to Success®. Denne er fremstilt i figur 10.4.

Trinn 1: vurdering (assessment).

Man starter med å vurdere virksomhetens mål og visjon, hvilke utfordringer den står ovenfor, og hvilke muligheter den har. Videre etableres en plan for forandringsarbeidet.

Trinn 2: Strategi

Her arbeider man videre med virksomhetens strategi med fokus på kundenes behov.

Trinn 3: Mål (Objectives)

Her bryter man ned de delene av strategien man arbeidet med i trinn 2 til strategiske mål. Dette er selve byggesteinene for strategien. Målene ordnes etter perspektiv og dokumenteres etterpå i etter årsakssammenhenger i tematiske strategikart. Et slikt er vist i figur 8.6.

Trinn 4: Strategikart

De tematiske strategikartene slås sammen til virksomhetsdekkende strategikart. Disse skal vise hvordan virksomheten skaper verdier for kunder og eiere.

Trinn 5: Ytelsesmåling

Her skal ytelsesmålinger for hvert av de strategiske mål utvikles. Hva skal måles, og hvilke verdier bør de ha?

Trinn 6: Initiativer

Nå skal man utarbeide initiativer for å støtte de strategiske målene. Disse må også få støtte i organisasjonen, og de må ha et eierskap. Initiativene dokumenteres.

Trinn 7: Automatisering

Her skal man få på plass programvare for å måle ytelse. Målekortene implementeres i denne programvaren, for eksempel i form av dashbord. Dette skal gi rask tilgang til data og dermed hjelpe beslutningstagere med å ta bedre beslutninger. I første omgang er det på toppledernivå dette skjer.

Trinn 8: Spre nedover i organisasjonen (Cascade)

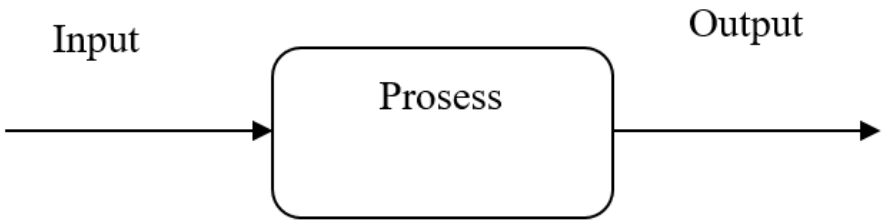
Tilgangen til og bruken av målekort skal nå spres nedover i organisasjonen. Det betyr at de stadig må detaljeres videre, og nye brukere må få tilgang til dem.

Trinn 9: Evaluering

I dette trinnet skal effekten av målekortene evalueres. Ser vi de forventede resultater? Måler vi de riktige tingene? Har det skjedd endringer i omgivelsene som vi må ta hensyn til? Dette er en kontinuerlig prosess, som før eller senere fører til at man begynner forfra igjen.

7.4 Hva skal vi måle?

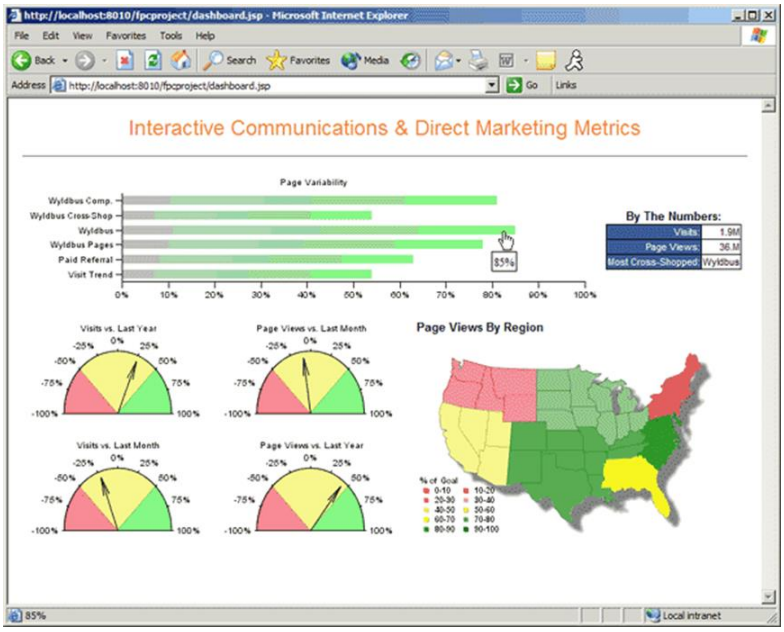
Tradisjonelt har man styrt bedriften etter en rekke enkeltmål, som fortjenestemargin, omsetningshastighet, feilprosent, svinn osv. Dette er relativt enkle mål å gjøre, og brukere er godt vant med dem. Målingene er i dag typisk knyttet til prosessene, som vi altså kan tegne slik:



Det er lettest å måle output fra en prosess, og dette er kanskje det vanligste enkeltmål. Output kan for eksempel være antall av en vare som blir produsert i prosessen, eller antall hjelp-samtaler som blir utført av en helpdesk-prosess. Med moderne ledelsesmetoder tilstreber man gjerne å gi de som utfører prosessen frihet til selv å utføre arbeidet best mulig, og da kan det være nok å måle output.

Ofte vil man imidlertid kombinere måling av input og output, for eksempel for å finne svinn. Andre ganger kan det være nødvendig å opprette "målepunkter" inne i selve prosessen, iallfall i forbindelse med prosessforbedringer.

En tendens i dag er å bruke dashbord til å presentere et lite antall indikatorer, som man så bruker ut fra prinsippet om målstyring. Typisk er at indeksene presenteres i et dashbort, gjerne som målevisere med en bakgrunn som går fra rødt via gult til grønt. Rødt viser at man er utenfor ønsket verdi for indeksen, gult at man er i faresomen og grønt at man er innenfor ønskede verdier. Et typisk dashbord er vist i figur 7.9.



Figur 7.9 Et typisk dashbord (Pentaho BI Server)

Dashbordet over viser nøkkeltall. I dag er det imidlertid mange som anbefaler å lage egne indekser som gir et samlet bilde basert på mange variabler.

Et eksempel på hvordan man lager en indeks er gitt av Mark G. Brown (Brown 2007). Det er basert på noe mange kjenner godt til: vår egen helse. på nettet finnes en rekke steder der man kan få en indikasjon på egen helse ved å svare på en rekke spørsmål. For en lege inngår også en rekke prøver, som blodtrykk, kolesterol m.m.

Når man skal bygge opp en slik indeks, må man ha en idé om hvilke faktorer som er av betydning, og hvor viktig hver enkelt av dem er (ikke alle er like viktige for helsen). Slike verdier kan inkludere:

- alder
- blodtrykket
- kolesterol
- jern
- familiehistorie
- egen sykehistorie
- spisevaner
- alkoholforbruk
- røyking
- mosjon
- osv

For å lage en helseindeks må man velge hvilke av disse som har mest innvirkning på helsen, og vekte hver av dem. Et mulig oppsett for å lage indeksen er å gruppere utvalgte variabler.

Historie 15 %	Helsedata 50 %	Livsstil 35 %
Familiehistorie 40 %	Kolesterol 20 %	Kosthold 30 %
Personlig historie 60 %	Blodtrykk 20 %	Mosjon 25 %
	Hjertets helse 20 %	Røyking 25 %
	Blodkjemi 20 %	Stress 15 %
	Vekt/høyde/alder 20 %	Medisinbruk 5 %

Tabell 7.2 En vektet liste over helsefaktorer(ikke nødvendigvis medisinsk korrekt)

Ut fra denne listen kan man nå beregne en total helseindeks. Denne må sammenlignes med ideelle verdier for indeksen.

Helseindeksen peker også på noen ytterligere problemer med målinger: noen av dem kan vi gjøre noe med, andre ikke. For eksempel har arv (familiehistorie) betydning for risikoen

for å utvikle visse sykdommer. Dette er imidlertid ikke noe vi kan gjøre noe med (før genterapi er i allmenn bruk). Andre verdier kan måles objektivt, som de i kolonnen for helsedata. Når det gjelder livsstil, er det imidlertid gjerne stor usikkerhet knyttet til hva pasienten oppgir. Folk flest har en tendens til å overdrive hvor mye de mosjonerer, og "underdrive" hvor mye de spiser, drikker og røyker. Dette er problemstillinger som også gjelder for målinger i en bedrift.

Hvis nå helseindeksen presenteres i et dashbord, kan vi drille ned for å studere nærmere detaljene bak indeksen. Dette er først og fremst aktuelt dersom indeksen avviker fra idealområdet.

Vi kan nå føre det samme prinsippet over til målinger i en bedrift. La oss si at vi ønsker en indikator på hvor tilfredse de ansatte er. Vi starter med å velge variabler, for så å vekte dem. Et eksempel kan være slik:

Mål	Skala	Vekt
Turnover i %	1 - 100	20 %
Stressfaktor	1 - 100	30 %
Spørreundersøkelser	1 - 100	40 %
Jobbsøkere	1 - 100	10 %

Tabell 7.3 Mulig bruk av variabler for ansattes tilfredshet

Vi kan ta et eksempel til på finansielle mål. En indeks for finansiell status kan inkludere flere målinger:

- Netto inntjening
- Ordrereserver
- Utestående fordringer
- Kapital bundet i lager

Dette er bare eksempler. Den enkelte bedrift må selv lage sine skorekort med vektinger. Endelig er det hele basert på begrenset rasjonalitet-modellen for beslutninger. Dersom den virkelige styringen avviker sterkt fra dette (særlig vanlig er politiske beslutninger), hjelper det ikke at man har et godt system for målinger.

Endelig: dataene som brukes i dashbordet hentes fra datavarehuset.

Kapittel 8 Introduksjon til datavarehus

Som vi tidligere har sett, trenger vi en egen database for systemer av typen MIS, BSS og EIS, som henter data fra systemene for daglig drift. Et datavarehus – engelsk Data Warehouse – er en slik database. I begrepet datavarehus ligger imidlertid mer enn at det bare er en database.

8.1 Definisjon

Begrepet Data Warehouse oppstod på slutten av 1980-tallet. Den første artikkelen som beskrev en arkitektur for datavarehus ble publisert i en artikkel i IBM Systems Journal i 1988 av Devlin og Murphy (Devlin 1988), som arbeidet ved IBMs forskningavdeling. Ideene i denne artikkelen ble siden videreutviklet av William Inmon, også ved IBM's forskningsavdeling. Han publiserte sitt arbeide i en klassisk bok fra 1992, som senere er kommet i nye utgaver (Inmon 1996). Inmon definerte et Data Warehouse som ***"a subject-oriented, integrated, time-variant, and non-volatile collection of data in support of management's decision-making process"***. Vi skal kort forklare de fire grunnleggende elementene i denne definisjonen:

- Subjektorientert (Subject-oriented): Et subjekt er høynivå entitetstyper i virksomheten, eller som virksomheten vil ha data om. Dette kan være kunder, leverandører, varer, ansatte, studenter eller forelesere. Dette i motsetning til et TPS, som typisk er orientert mot funksjoner, som fakturering, lagerstyring eller avlønning. Subjektene i datavarehuset kaller vi også ***dimensjoner***.
- Integrert (integrated): Data fra ulike kilder samles i en felles database, der de også gis ***felles definisjoner*** (dato eller kunde er for eksempel likt definert uansett hvordan definisjonen er i systemene data hentes fra)
- Tidsvariabel: Et datavarehus viser historiske data, det vil si at vi alltid skal kunne se data i forhold til ***tid***. I datavarehuset er også tiden en dimensjon.
- Ikke-flyktig (nonvolatile): Når data er lest inn i datavarehuset, skal de ikke forandres på. Dette i motsetning til i et TPS, der visse data forandrer seg kontinuerlig (banksaldo, lagerbeholdning). Dataene skal heller ikke slettes (før den tid kommer da de ikke lenger er interessante). En annen betegnelse for det samme er ikke-oppdaterbar (nonupdatable) (Hoffer 2002).

Inmons definisjon av et datavarehus fokuserer på karakteristika ved dataene som lagres. Andre definisjoner utvider definisjonen ved også å ta med behandling av data fra de hentes ut fra andre datasystemer til de presenteres for brukerne. Uansett definisjon er målet med et datavarehus å integrere data fra forskjellige deler av virksomheten i en felles database, der brukere enkelt kan få tilgang til dem gjennom spørringer, rapporter og analyser (Connolly 2002).

Problemet for mange virksomheter er at de nærmest drukner i data, men allikevel mangler informasjon (Hoffer 2002). Dette paradokset skyldes at data ligger i databasene til transaksjonsprosesseringssystemene, noe som gjør at det er vanskelig å gjøre dem om til informasjon. En database laget for et transaksjonsprosesseringssystem er uegnet som

database for et datavarehus fordi det er helt forskjellige krav til de to systemtypene. Databasen til et TPS skal være optimal for behandling av transaksjoner, mens databasen i et datavarehus skal være optimal for behandling av spørringer, spesielt *ad hoc*-spørringer. De forskjellige egenskapene er vist i tabell 9.1 (Connolly 2002). Det å gjøre data fra transaksjonsprosesserings-systemene om til informasjon for ledere kalles gjerne "å bygge bro over informasjonskløften" ("bridging the information gap").

Transaksjonsprosesseringssystemer	Datavarehus
Inneholder aktuelle data	Inneholder historiske data
Inneholder detaljerte data	Inneholder detaljerte, lett summerte og sterkt summerte data
Dynamiske data	Statiske data
Repetitiv, strukturert prosessering	Ad hoc, ustrukturert og heuristisk prosessering
Mange transaksjoner	Lite transaksjoner
Forutsigbar bruk	Uforutsigbar bruk
Transaksjonsdrevet	Analysedrevet
Applikasjonsorientert	Subjektorientert
Støtter dag-til-dag beslutninger	Støtter strategiske beslutninger
Betjener et stort antall operative brukere	Betjener et relativt lite antall ledere

Tabell 8.1 Forskjeller mellom TPS og datavarehus

En virksomhet kan ha mange forskjellige transaksjonsprosesseringssystemer med hver sine databaser (men med foretakssystemer er det en enkelt, integrert database). Virksomhetens transaksjonsprosesseringssystemer støtter operativ drift, som salg, fakturering, regnskap, lønnsberegning, lagerstyring, produksjonsplanlegging m.m. Databasen er typisk sterkt normalisert for å støtte transaksjonsbehandlingen, og kan også på andre måter være tunet for å få mest mulig effektivitet. Rapportering på slike systemer innskrenkes til enkle rapporter som kan brukes i dag-til-dag beslutninger, mens ad hoc-spørringer og kompliserte analyser normalt ikke tillates. Dette er fordi disse systemene er kritiske for den daglige drift, og derfor ikke må forstyrres med andre oppgaver. En ad hoc-spørring mot en normalisert database av denne typen kan bli svært komplisert, og man vil ha begrensede muligheter til å beregne på forhånd hvor mye av systemets ressurser den vil kreve. Siden transaksjonsprosesseringssystemer vesentlig inneholder aktuelle data, er det også begrenset hva slags analyser man kan bruke dem til.

Et datavarehus vil inneholde data fra hele virksomheten, og disse data akkumuleres over tid. Datavarehuset vil dermed inneholde historiske data, noe som gjør at man kan gjøre en rekke analyser der tiden er viktig (eksempel sammenligne salg for september i år med september i fjor). Siden datavarehuset ikke er kritisk for den daglige operative virksomheten, gjør det ikke så mye om man setter i gang en spørring som krever svært lang tid (det kan selvsagt irritere andre brukere, men det er ikke kritisk). Ikke minst kan man optimalisere databasen for spørringer i stedet for transaksjonsbehandling. Dette

gjør det enklere å lage spørringer, og de blir mer forutsigbare. Datavarehuset kan også inneholde ferdige summeringer av data.

De overordnede målene med et datavarehus er knyttet til ledelsens beslutninger (Connolly 2002). Følgende fremheves gjerne som argumenter for å utvikle en datavarehusløsning:

- Økt produktivitet hos virksomhetens beslutningstagere: Datavarehuset øker produktiviteten til beslutningstagere ved at det inneholder informasjon i form av integrerte, konsistente, subjektorienterte historiske data fra forskjellige virksomhetsområder.
- Konkurransefordeler: Det er dokumentert hvordan virksomheter oppnår konkurransefordeler ved at beslutningstagerne får tilgang til informasjon som tidligere var skjult i datamengdene; informasjon om kunder, salgsområder, produkter m.m.
- Potensiell stor avkastning på investeringen: Datavarehusprosjekter er svært kostbare. Det er imidlertid dokumentert hvordan mange virksomheter har fått mangedobbelt tilbake av investeringene i datavarehus. En undersøkelse konkluderte for eksempel med at over 90 % av de virksomheter som investerte i datavarehus hadde en avkastning på over 40 % på denne investeringen, noen av dem mye mer enn dette. Disse avkastningene har sammenheng med de to punktene foran.

Det er gjort noen undersøkelser av hva som er kritiske suksessfaktorer ved innføring av datavarehus. I en slik undersøkelse (Wixom 2001) var en konklusjon at kvaliteten på data i datavarehuset var en svært viktig suksessfaktor, ved siden av kvaliteten på systemet i seg selv.

8.2 Datavarehusets arkitektur

Et datavarehus er imidlertid ikke bare selve databasen. Det er mer det vi kan kalle et konsept, bestående av selve databasen, verktøy for å hente ut data fra andre informasjonssystemer, prosesser for klargjøring av data som skal lastes i databasen, og verktøy for å rapportere på dataene.

Et datavarehus henter alltid sine data fra andre systemer i organisasjonen, typisk forskjellige transaksjonsprosesseringssystemer. Disse systemene kaller vi gjerne *kildesystemer*. Dataene må først hentes ut av kildesystemene, deretter behandles på forskjellige måter før de kan lastes inn i datavarehuset. Denne behandlingen, som forbereder dataene før lasting, kalles ETL (Extraction, Transformation, Loading), og de samlede prosedyrer og datalagring involvert i klargjøringen kalles Data Staging Area. De klargjorte dataene lastes så inn i selve databasen. Forskjellige typer verktøy brukes så til rapporteringen på denne. Vi kan dermed operere med fire forskjellige elementer i datavarehuset, som vist i figur 8.1. (Sharda 2017). Dette kalles gjerne datavarehusets arkitektur.

Fra kildesystemene hentes altså data. En viktig del av utviklingen av et datavarehus er å kartlegge hvilke data som finnes i hvilke systemer. Disse data hentes ut ved hjelp av spørreverktøy for den typen datalagring som brukes av kildesystemet. Er dette basert på en relasjonsdatabase, kjøres en SQL-spørring på kildesystemets maskin, er det flate filer på en OS/400-maskin kjøres en RPG III-rapport osv. For ar resultatet av en slik spørring skal kunne brukes i Data Staging Area, er det vanlig å legge dem ut som flate filer for overføring til maskinen der DSA befinner seg (typisk en egen server).

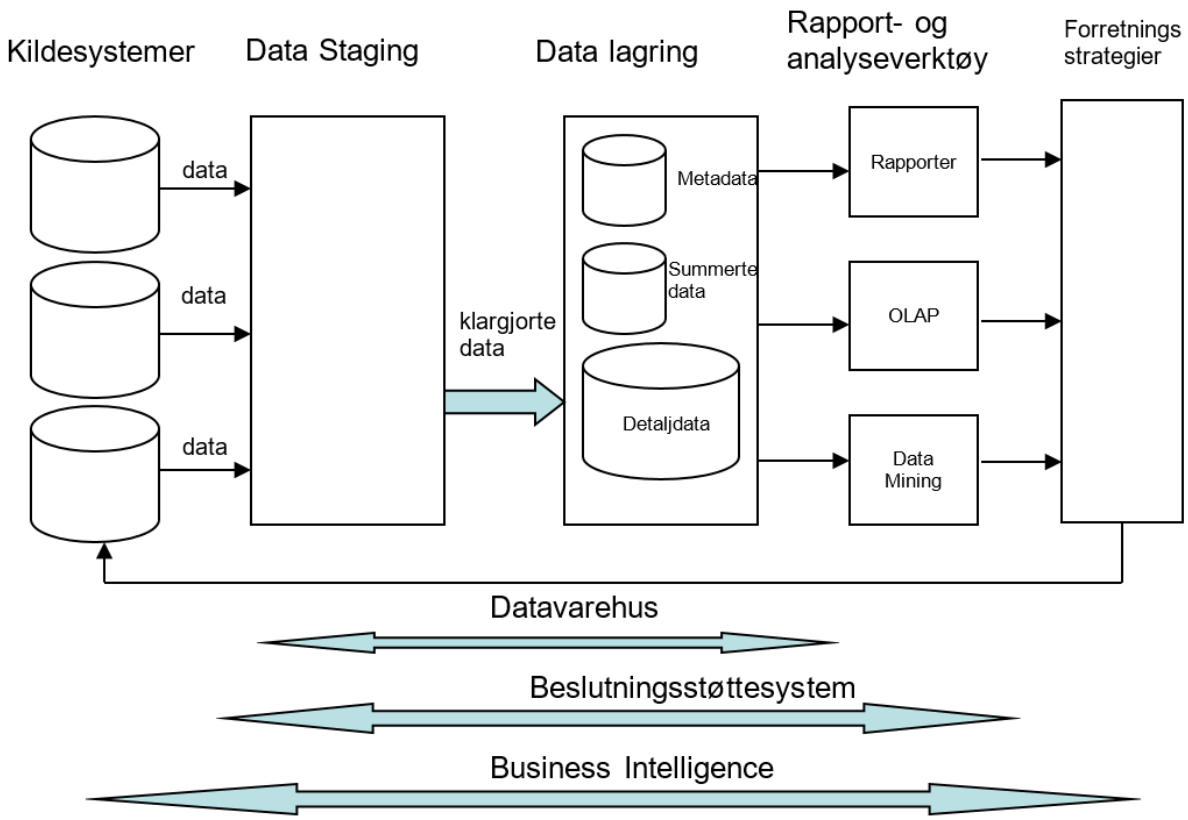
I DSA leses nå data hentet fra kildesystemene, og de mellomlagres. Dette skjer ofte som flate filer, fordi slike er mye mer ressurseffektive å operere på enn databasetabeller. Typiske oppgaver i DSA er rensing av data (det vil alltid være feil i datagrunnlaget), fjerning av duplikater, omregninger, standardiseringer, sorteringer, summeringer med mer (vi skal se nærmere på dette i et eget kapittel). Til slutt vil de ferdig behandlede data bli lastet inn i datavarehuset.

Selve datavarehuset vil inneholde data med forskjellig detaljnivå. Noen data kan være rene transaksjoner, som et bestemt uttak av en vare fra et lager eller et bestemt salg. Andre data kan være mer eller mindre summert, som for eksempel sum salg av en bestemt vare for en bestemt uke. Uansett vil de data som ligger i datavarehuset være historiske data, i motsetning til data i TPSenes databaser.

En viktig type data i datavarehuset er metadata. Dette er ”data om data”, og er sentrale for bruken av datavarehuset.

Sluttbrukerverktøy for bruk mot datavarehuset kan være av mange typer. Først og fremst brukes SQL for spørringer mot tabellene i datavarehuset, men vanligvis ikke ved at brukerne selv skriver SQL-spørringene. I stedet genereres SQL-statements av verktøyene som brukes. Brukerne vil ha tilgang til spørreverktøy og rapportgeneratorer, men også til mer spesialiserte analyseverktøy som gir mulighet for avanserte analyser. Eksempler på slike er verktøy for OLAP (OnLine Analytical Processing) og Data Mining. Vi skal se nærmere på disse i et eget kapittel.

Kimballs (Kimball 2013) arkitektur for datavarehuset er mye brukt som referansemodell. Det finnes imidlertid varianter over denne som er vel så interessante. Giovinazzo (2000) har en lignende modell som kombinerer datavarehus, beslutningsstøttesystem og Business Intelligence. Modellen er vist i figur 9.2. Giovinazzo begrenser datavarehuset til Data Staging og datalagring. Datavarehuset pluss sluttbrukerverktøyene utgjør et beslutningsstøttesystem. Når output fra beslutningsstøttesystemet brukes av forretningsstrateger, som så vil initiere tilpasninger i kildesystemene (for bedre å oppfylle bedriftens informasjonsbehov), har vi en Business Intelligence *sløyfe*.



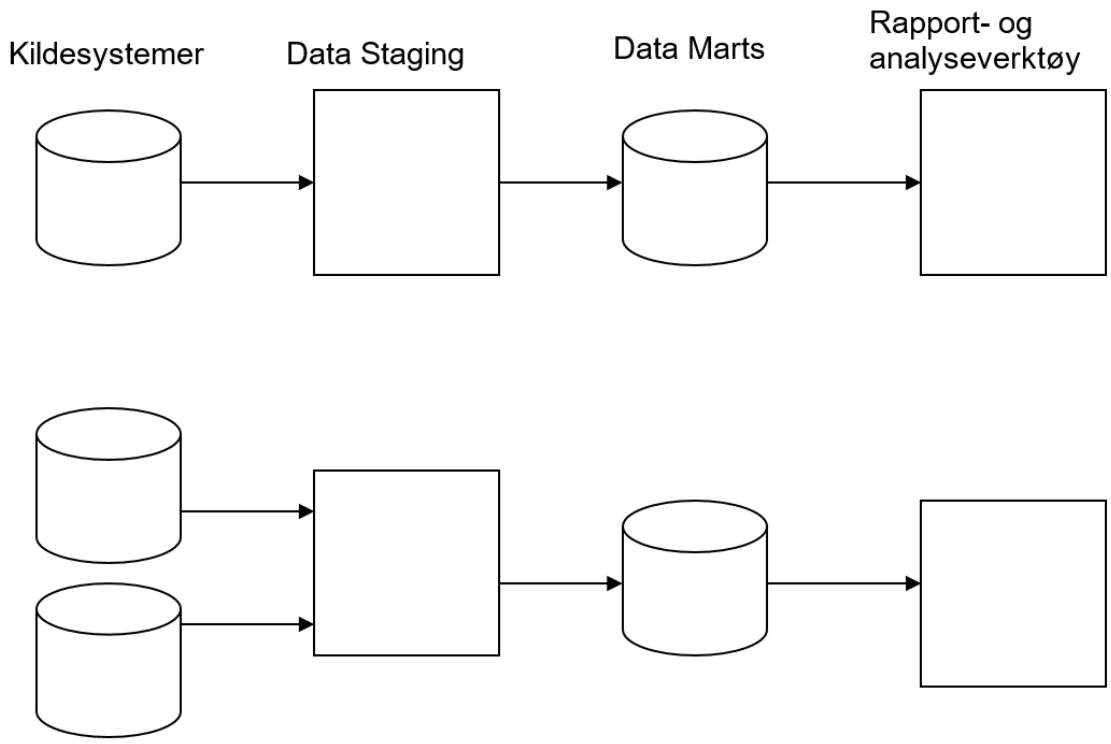
Figur 8.1 Business Intelligence sløyfen (etter Giovinazzo 2000)

I den videre fremstillingen skal vi bruke Kimballs arkitekturmodell. Det er imidlertid enkelt å bruke Giovinazzos BI-sløyfe etter de samme prinsipper.

8.3 Data Marts

En viktig del av datavarehusets arkitektur er såkalte Data Marts (”datamarkeder” på norsk, men jeg har aldri hørt noen bruke dette ordet). Det finnes forskjellige oppfatninger av hvordan et Data Mart skal defineres og hvilken rolle de spiller i datavarehuset. Vi skal se nærmere på denne diskusjonen, men først skal vi se på hvor begrepet stammer fra.

Før datavarehusbegrepet oppstod, hadde mange virksomheter laget mindre databaser der de samlet informasjon for spesifikke forretningsområder. Det kunne for eksempel være en database for salgsinformasjon, som hentet data fra TPSet der salg ble registrert. Da det utover 1990-tallet ble populært å utvikle datavarehusløsninger, der en grunnleggende ide altså er å integrere data fra forskjellige systemer, viste det seg at dette var kompliserte og langvarige prosjekter. I mange virksomheter ble brukerne lei av å vente på at det sentrale datavarehuset skulle bli ferdig, og utviklet en egen løsning skreddersydd sine egne behov. Slike ”minidatavarehus” ble etter hvert kalt data marts.

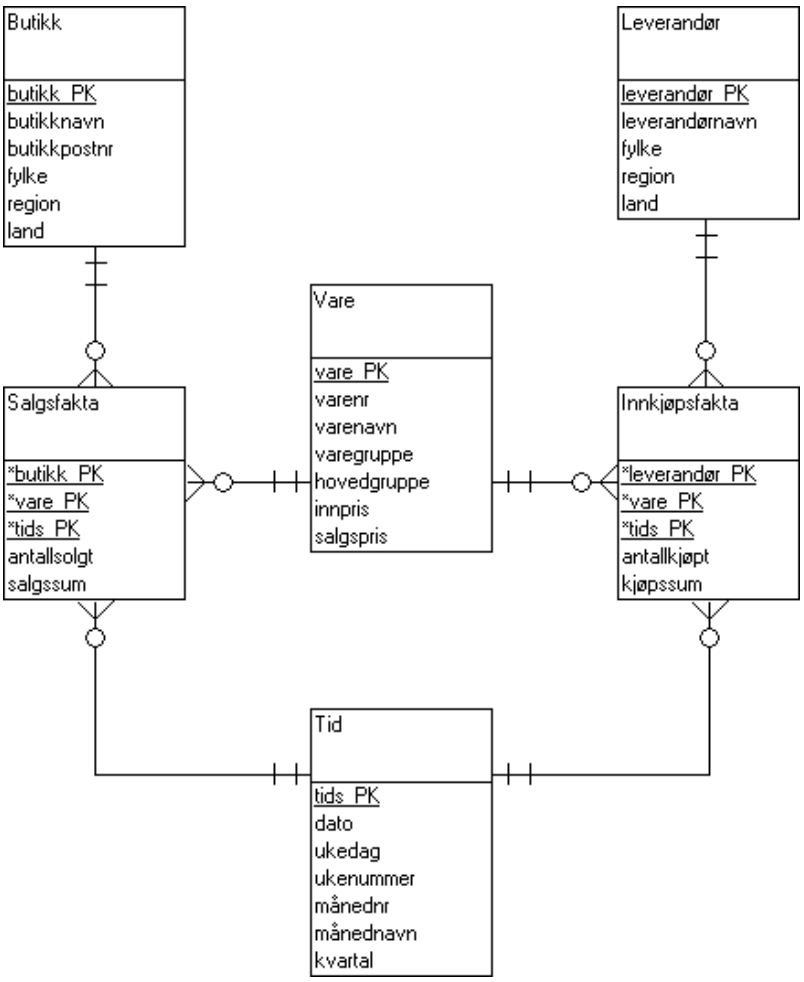


Figur 8.2 To uavhengige data marts, med henholdsvis ett og to kildesystemer

Ideen om å ha minidatavarehus skreddersydd til behovene i en bestemt del av virksomheten viste seg imidlertid å være fruktbar. Det er sjelden beslutningstagerne innenfor en del av virksomheten trenger informasjon om andre deler av virksomheten, så det er unødvendig å lage et system der alle har tilgang til alt. Et slikt system vil også bli unødig komplisert å bruke. I dag hører derfor data marts med i alle datavarehusløsninger.

Det er to ”skoler” når det gjelder definisjonen av data marts og hvilken rolle de skal spille i utviklingen av et datavarehus.

1. Selve datavarehuset er bygget opp av uavhengige data marts. Datavarehuset blir dermed unionen av alle data marts (Kimball 2013).
2. Alle data marts er avhengige, det vil si at de får sine data fra selve datavarehuset. Mens datavarehuset først og fremst inneholder detaljdata, vil data marts først og fremst inneholde summerte data for spesifikke virksomhetsområder (artikkel i DBMS Magazine).



Figur 8.3 Datavarehusbuss (Etter Kimball 2013)

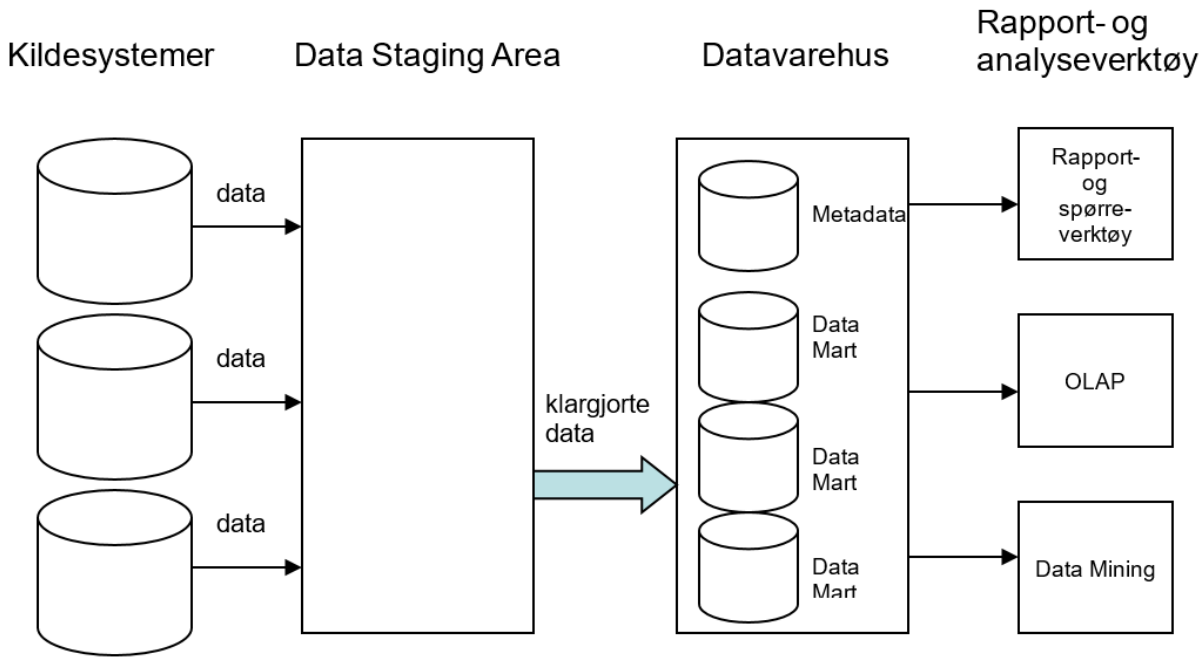
Kimball argumenterer for å bygge opp et datavarehus ved å lage ett og ett data mart. Disse vil "henge sammen" ved at man bruker felles datadefinisjoner og dimensjonstabeller. Kimball kaller denne standardiseringen datavarehusbussen (en buss er et standardisert grensesnitt). Selv om hvert data mart er uavhengig, vil de allikevel utgjøre en helhet (datavarehuset) på grunn av standardene.

Figur 8.3 viser hvordan de samme dimensjonene kan brukes av to forskjellige faktatabeller i en butikkjede. Faktatabellen Innkjøpsfakta viser grossistleddets innkjøp av varer fra forskjellige leverandører, mens Salgsfakta viser butikkenes salg av forskjellige varer. Dimensjonene Tid og Vare er felles for de to faktatabellene.

Også i Inmons modell er standardisering av dimensjoner og datadefinisjoner en sentral egenskap ved datavarehus og data marts.

Selv om Kimball er en av de store guruene på datavarehus internasjonalt, er det mange som kritiserer hans modell for oppbygging av datavarehus ved hjelp av uavhengige data marts, blant dem Inmon (Inmon 1997, 2000). Kritikerne går i stedet for avhengige data

marts, altså data marts som henter sine data fra et integrert datavarehus. Dette ser også ut til å være den vanlige modellen for oppbygging av datavarehus, iallfall her i landet.

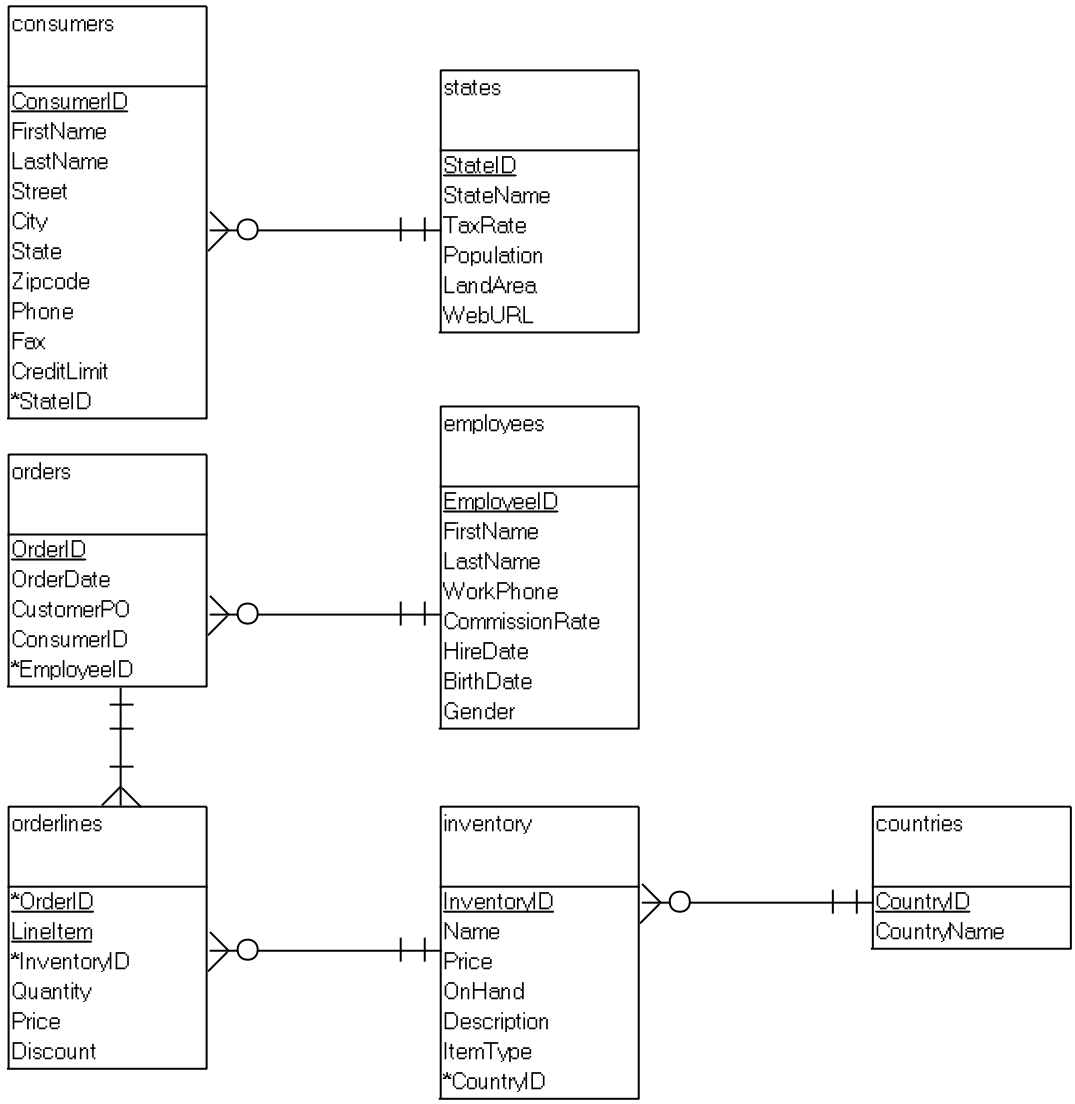


Figur 8.4 Kimballs datavarehusmodell basert på uavhengige data marts

Beskrivelse av eksempelbedrift

Vi skal nå bruke en eksempelbedrift for å illustrere prinsippene for datavarehus og Business Intelligence. I første omgang skal vi opprette en kildedatabase. Tabellene med data er hentet fra CD'ene som følger boken *Introduction to Oracle 10g* av James Perry og Gerald Post (Pearson Prentice Hall 2007).

Tabellene er databasen til en nettbutikk som selger kaffe og te til kunder over hele verden. Databasen ser slik ut:



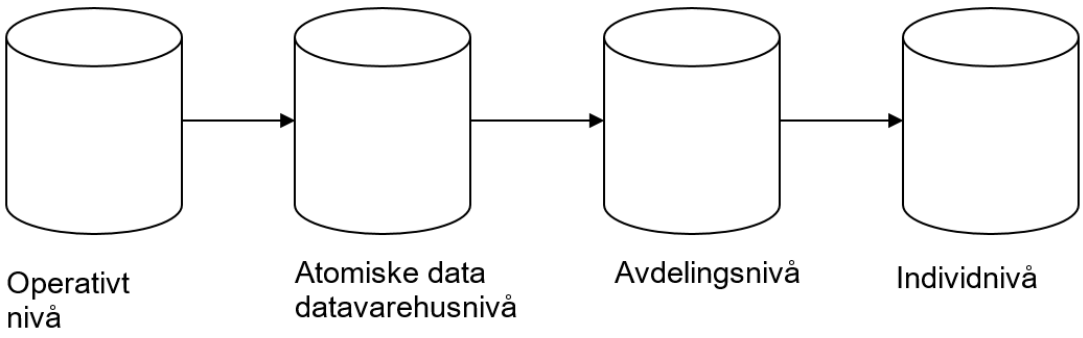
Bedriften har et stort utvalg i kaffe- og tesorter dyrket i en rekke land. Det er 7 tabeller i databasen. Kundetabellen inneholder mer enn 1500 rader. Bedriften har 22 ansatte som behandler ordre, både fra nettbutikken og over telefon. Ledelsen ønsker å måle hvor mye den enkelte ansatte selger for, så hver ordre er knyttet til en ansatt. Telefonsalg registreres på den ansatte som mottar ordren, mens ordre fra nettbutikken tildeles ansatte etter et rotasjonsprinsipp (en stor del av de amerikanske ordrene kommer via telefon). Videre ønsker man å vite hvordan de enkelte merkene selger, hvilke kunder som kjøper mest og hvor de kommer fra. Alle varene har en listepris, men selgeren kan gi en rabatt ved bestillingen.

Kapittel 9 Modellering av datavarehuset

Fra datamodelleringen og databasekurset vet vi at en datamodell skal normaliseres til Boyce-Codd normalform (BCNF). Hensikten med dette er å sikre oss mot redundans og inkonsistens ved transaksjoner, der det skjer oppdateringer av databasen. Normaliserte datamodeller er altså først og fremst beregnet for transaksjonsprosesseringssystemer. Prisen vi må betale for normaliseringen er at SQL-spørringer blir kompliserte. I et datavarehus skal imidlertid ikke data oppdateres når de først er lest inn i databasen. Derfor kan man i stedet prioritere optimalisering med hensyn på spørringer. Det er innen datavarehusverdenen to "skoler" når det gjelder modellering av datavarehus. Den ene "skolen" hevder at modellering av datavarehus skal følge de samme grunnleggende prinsipper som for databaser ellers, mens den andre står på at datavarehus skal bygges på en egen type modeller kalt dimensjonsmodeller. William Inmon og Chris Date er sentrale eksponenter for den første "skolen", mens Ralph Kimball er det sentrale navnet innen den andre.

9.1 Inmon om modellering

Mark Inmon er mannen bak datavarehuskonseptet, og beskriver i sin bok (Inmon 1996) hvordan et datavarehus bør modelleres. De samme prinsippene støttes av Chris Data (Date 2003). Inmon beskriver datavarehusarkitekturen som i figur 10.1.



Figur 9.1 Inmon's datavarehusarkitektur

Inmon bygger på at det er to grunnleggende forskjellige typer data i datavarehuset: primitive og avledede. De primitive data er de som brukes i den daglige drift, som transaksjonsdata, mens avledede er summerte eller beregnede data. Disse er sammenlignet i tabell 10.1.

Primitive data	Avledede data
Kan oppdateres	Oppdateres ikke
Aktuelle verdier	Historiske verdier
Opereres på av repetitive prosedyrer	Heuristiske, ikke repetitive prosedyrer
Operative data, brukes i daglig drift	Beslutningsdata, brukes av ledelsen
Ikke nødvendigvis integrert	Integrert

Tabell 9.1 Sammenligning av primitive og avledede data

Inmon opererer med tre nivåer på datamodeller:

- Høynivå: En virksomhetsmodell (Corporate Data Modell) som er kombinasjonen av mange forskjellige views av virksomheten (salg, lønning, finans m.m.). Viser entitetstyper og relasjoner mellom disse. Dette er det samme som en konseptuell modell for hele virksomheten.
- Mellomnivå; En Data Item Set-modell med primærnøkler, fremmednøkler, attributter og datatyper for disse. Dette er den typen datamodell vi er vant med å utvikle i Modelator, og som kan eksporteres til et databasescript.
- Lavnivå: En fysisk modell som gir selve tabelldefinisjonene med indekser. Her kommer også andre ting inn, som blokkstørrelser. Målet med den fysiske modellen er å økonomisere på I/O, siden I/O er langsommere enn behandling i internminnet.

I praksis kan datamodellen slik Inmon foreskriver den ligne på modellen fra kildesystemet. Noen vesentlige forskjeller er det imidlertid:

- Tabeller i kildesystemet som ikke er relevante for beslutningsstøtte er ikke med
- Attributter som er nødvendige i operativ bruk, men som ikke er nødvendige for beslutningsstøtte, er fjernet
- Alle data har en ”timestamp”, som ofte inngår i sammensatte primærnøkler
- Ofte er det allerede gjennomført JOIN av tabeller

Inmon åpner imidlertid også for bruk av det han kaller star joins. Dette er modeller der numeriske data ligger i en tabell, koblet til tabeller med hovedsakelig tekstlige, beskrivende data. Dette er den typen modell vi i kapittel 9.2 skal beskrive nærmere under navnet dimensjonsmodell. Inmon foreslår bruk av en kombinasjon av normaliserte entity-relationship modell og star joins.

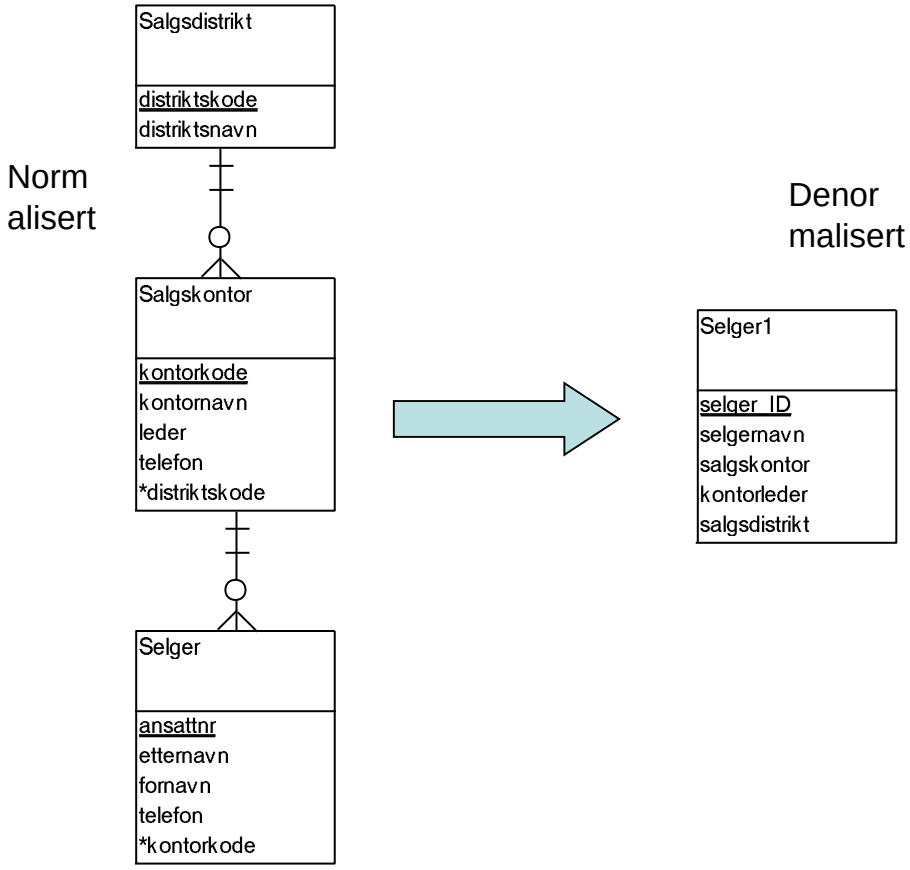
Chris Date er mye mer en ”ER-purist” enn Inmon. Han er meget skeptisk til dimensjonsmodellering, og går så langt som å si at bruk av stjerneskjema (som er det samme som star join og dimensjonsmodell) er et tegn på at designeren har tatt snarveier i forhold til riktig designteknikk (Date 2003). Han påpeker imidlertid også at en skikkelig relasjonell design er svært lik et stjerneskjema, og at det ofte ikke vil være noen forskjell. Date mener også at star joins egentlig er fysiske modeller, og ikke logiske, selv om de av utviklerne behandles som logiske modeller. Fremfor alt mener han at den tilnærmingen til dimensjonsmodellering mange utviklere har, ikke er disiplinert nok, og at modellene derfor ikke er gode nok. Om denne kritikken egentlig gjelder dimensjonsmodellering i seg selv, eller enkelte utøvere av teknikken, er jeg ikke så sikker på.

9.2 Dimensjonsmodeller

Dimensjonsmodellering er en systematisk tilnærming til modellering av et datavarehus. Den sentrale talsmannen for denne type modellering er Ralph Kimball, lederen av et konsulentfirma i USA. Hans bok ”The Datawarehouse Lifecycle Toolkit” (Kimball 2013) gir en grundig behandling av dimensjonsmodellering. Innen datavarehusverdenen har det

lenge rast en debatt om hva som er den beste tilnærmingen til modellering av et datavarehus; dimensjonsmodellering eller ER-modellering. Det er imidlertid ikke nødvendigvis noen motsetning mellom disse to perspektivene. Agosta (Agosta 2000) argumenterer for at dimensjonsmodeller bare er en variant av ER-modellering, spesielt utviklet for beslutningsstøttesystemer.

En dimensjonsmodell består av en faktatabell koblet til flere dimensjonsmodeller. Faktatabellen er normalisert til 3. normalform, mens dimensjonstabellene er denormalisert til 1. normalform.



Figur 9.2 Denormalisering av dimensjon

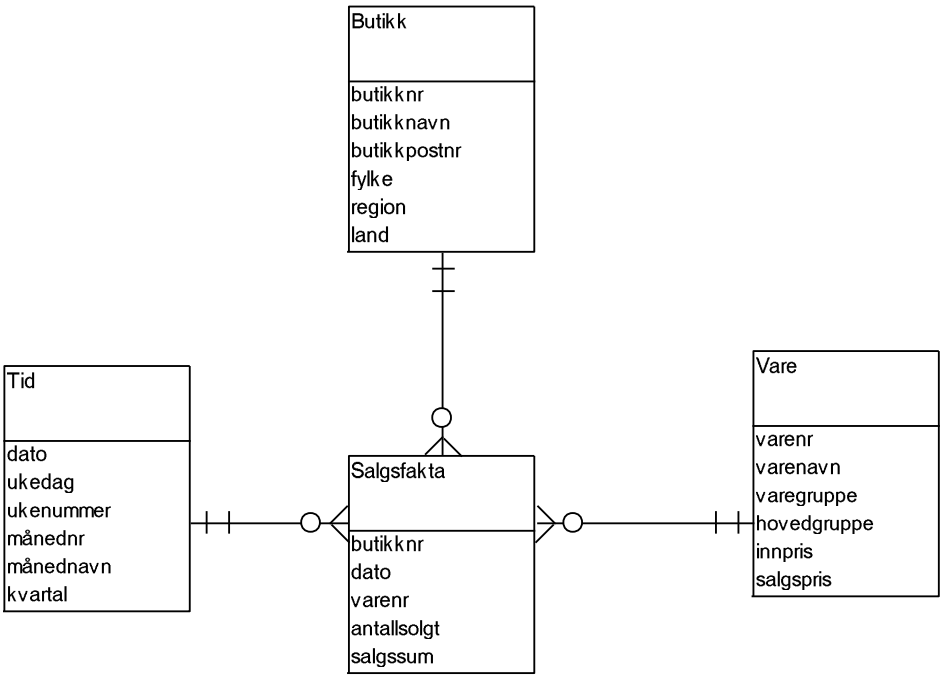
En dimensjon er hva som helst vi ønsker å vite noe om. Typiske dimensjoner er vare, kunde, leverandør, marked og ikke minst tid. Dimensjoner er altså det samme som subjektene fra definisjonen på datavarehus.

Dimensjonene kan ha mange attributter. Dimensjonen kunde kan for eksempel ha attributter som navn, adresse, postnummer, kundekategori osv. Dimensjonene vil dermed tilsvare våre vanlige entitetstyper i modellen. Typisk for dimensjonene er at de har attributter som inngår i et hierarki. I en vanlig ER-modell vil disse attributtene inngå i egne tabeller, mens de i dimensjonsmodellen er denormalisert til en tabell. Dette er vist i figur 9.2.

Det er viktig å merke seg at attributtene til dimensjonene er adskillig færre enn de vi vanligvis opererer med i et transaksjonsprosesseringssystem. I datavarehusets dimensjoner er vi bare interesserte i å ha med attributter som det er aktuelt å bruke som

kriterium for et utvalg. I praksis betyr dette at det ikke er relevant å ha med telefonnummeret til en butikk, mens derimot navnet må være med, og postnummeret kan også være interessant. I et datavarehus for et telekommunikasjonsselskap vil derimot telefonnummer være et selvsagt attributt.

Dimensjonene knytter vi til fakta, som er det vi kan måle. Fakta vil typisk være antall eller kroner (eller euro, dollar osv), som ligger i en egen tabell kalt en faktatabell. Typiske operasjoner på fakta er summering, opptelling og beregning av gjennomsnitt. Andre data, som for eksempel timelønn for selger, har ingen mening i faktatabellen. Slike data ligger som attributt i dimensjonstabellen. Alle dimensjonene er så knyttet direkte til faktatabellen. Dermed blir det et sort antall fremmednøkler i denne.



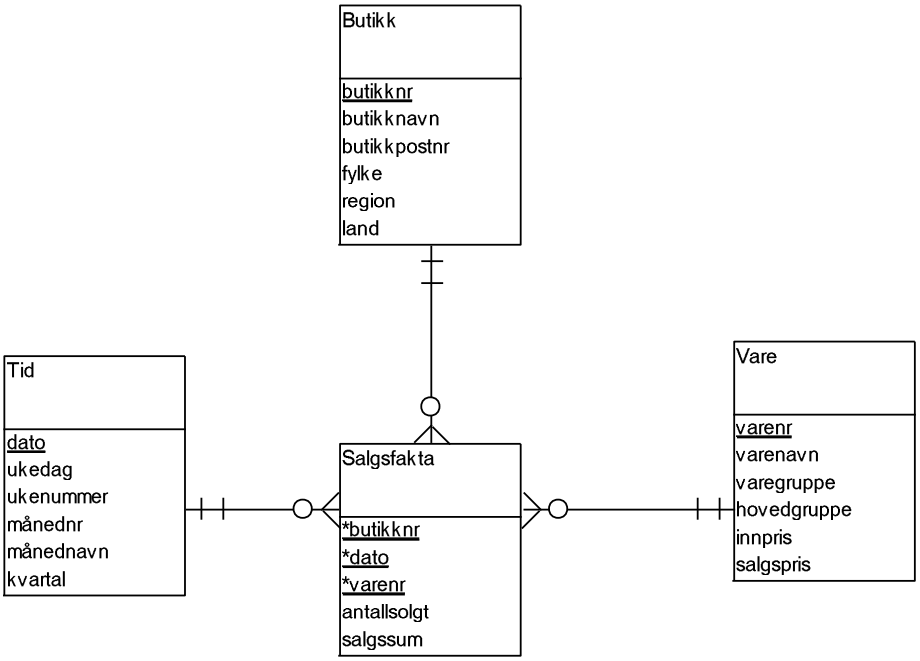
Figur 9.3 En enkel dimensjonsmodell for butikkjeden

Den normaliserte datamodellen representerer som regel hierarkier. Slike hierarkier forekommer ofte i den virkeligheten vi modellerer. I den denormaliserte modellen har vi ”mistet” hierarkiet, noe som kan være uheldig med tanke på senere informasjonssøk. I mange ETL-verktøy, for eksempel Oracle Warehouse Builder, kan vi imidlertid definere hierarkier i en denormalisert dimensjonstabell. Warehouse Builder bruker en tabell med metadata til å administrere dette.

I figur 9.3 er vist en enkel dimensjonsmodell for butikkjeden beskrevet i kapittel 13. Data i dimensjonstabellene i datavarehuset vil typisk hentes fra kildesystemene, med unntak av tidsdata

I modellen i figur 9.4 er brukt de samme primærnøklene i dimensjonstabellene som i kildesystemene. Foreløpig er det ikke vist primærnøkkel i faktatabellen. Vi kan her bruke to forskjellige grunnlag for primærnøkkel:

- En egen identifikator for hver post i tabellen. Dette kan være et transaksjonsnummer i de tilfelle det er transaksjoner fra kildesystemet som utgjør postene, eller et løpenummer som genereres i DSA
- En sammensatt primærnøkkel

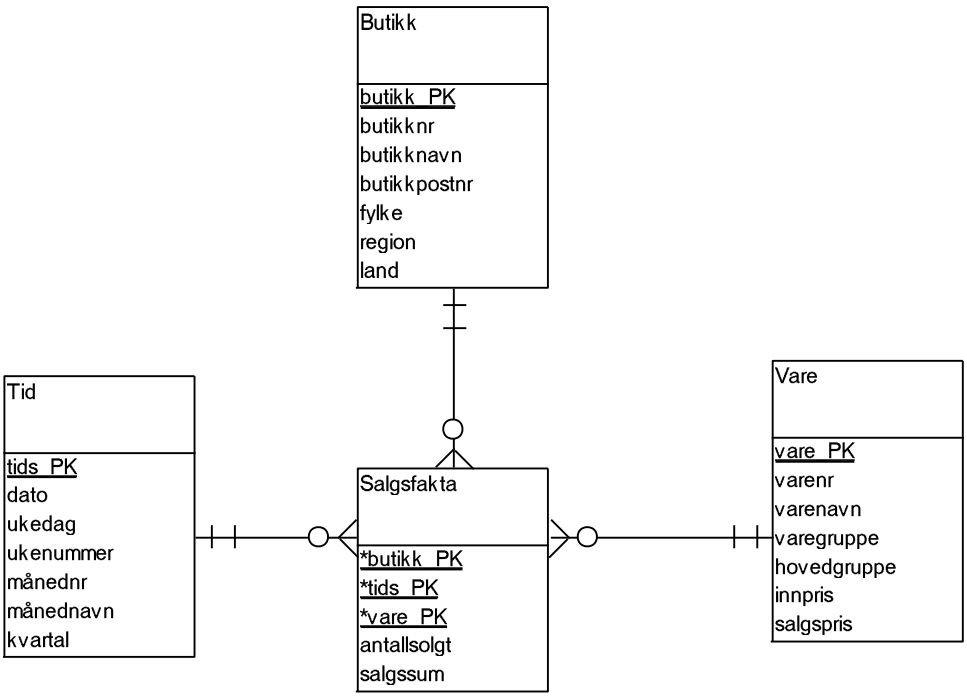


Figur 9.4 Dimensjonsmodellen med sammensatt primærnøkkel i faktatabellen

I dimensjonsmodellen i figur 9.3 og 9.4 er det hierarkier i alle tre dimensjonene. Dimensjonen Butikk har hierarkiet land -> region ->fylke -> butikk, dimensjonen Vare har hierarkiet hovedgruppe -> varegruppe -> vare, mens Tid har hierarkiet år ->kvartal -> måned -> uke -> dag.

Et mye brukt prinsipp for datavarehus er at man erstatter primærnøklene fra kildesystemene med surrogatnøkler. Noen mener dette er helt kritisk (Kimball 2013). En surrogatnøkkel er generert av datavarehuset i DSA, og sikrer at data i datavarehuset er robuste i forhold til fremtidige forandringer i varenummer, butikknummer, kundenummer m.m.. Slike forandringer er nemlig ikke uvanlige.

Med surrogatnøkler ser modellen vår ut som i figur 9.5.



Figur 9.5 Dimensjonsmodell med surrogatnøkler

Dimensjonsmodeller byr på mange fordeler i datavarehus (Kimball 2013). For det første er de et forutsigbart, standard rammeverk som kan brukes av spørreverktøy og brukergrensesnitt. Spørringer vil for eksempel ta utgangspunkt i attributter i dimensjonene. Videre er en slik modell nærmest uavhengig av mulige mønstre for spørringer mot databasen. Kimball kaller dette symmetri. En dimensjonsmodell er også svært fleksibel i forhold til å føye til nye fakta eller dimensjoner, eller legge til nye attributter i dimensjonene. Endelig gjør dimensjonsmodellen det mulig å håndtere typiske problemstillinger på en standardisert måte. Kimball nevner eksempler på slike problemstillinger:

- Langsomme forandringer i dimensjoner: Forandringer i attributtverdier hos dimensjoner som Kunde og Vare. Slike forandringer er både sjeldne og uforutsigbare.
- Heterogene produkter: Dette er situasjoner der virksomheten må behandle forskjellige prosesser sammen innenfor et samlet sett dimensjoner og fakta, men hvor forskjellige fakta ikke kan sammenlignes. Dette forekommer for eksempel i banker, som opererer med uttak og innskudd, sammen med saldoer på innskuddskonti og lånekonti.
- Hendelseshåndtering i databasen: Her skal systemet vite om hendelser det ikke nødvendigvis kan knyttes fakta til, som for eksempel at en student har søkt om opptak ved en høyskole. I slike tilfelle kan vi ha faktaløse faktatabeller.

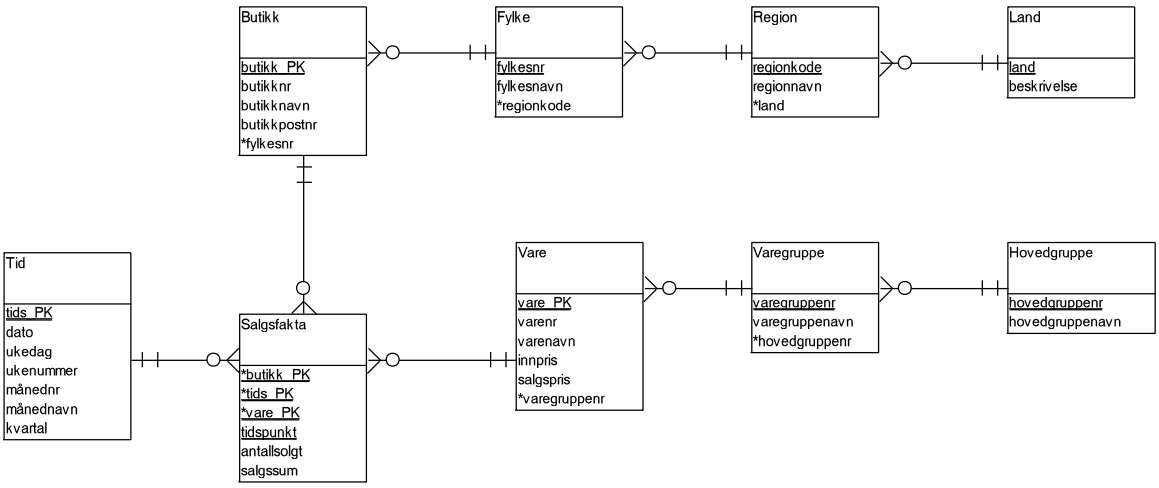
En siste styrke ved dimensjonsmodellen er at den muliggjør programvare for å håndtere aggregerte fakta. Aggregeringer betyr en planlagt redundans, det vil si at de samme data forekommer flere ganger i forskjellige summeringer. I et datavarehus er bruk av aggregerte data viktig ut fra effektivitet og ressursbruk. Programvare som gjør det mulig å navigere mellom ferdig aggregerte tabeller uten at brukeren merker det bygger på at disse tabellene inngår i dimensjonsmodeller.

Et datavarehus kan altså bestå av en rekke separate dimensjonsmodeller. Mange av disse vil bruke de samme dimensjonene. For å få det hele til å henge sammen i et datavarehus, er det viktig at vi standardiserer dimensjoner, fakta og attributter. Datavarehuset skal bare ha en enkelt kundedimensjon, uansett hvor mange forskjellige dimensjonsmodeller som bruker den. Datadefinisjoner i faktatabeller og dimensjoner skal også være standardiserte, slik at de betyr det samme i hele datavarehuset. Kimball kaller dette **datavarehusets bussarkitektur**.

Man skal allikevel være forsiktig med å la dimensjonsmodeller bli en universaloppskrift på hvordan et datavarehus skal modelleres. Poe, Klauer og Brobst (Poe 1998:208 ff) sier at dimensjonsmodellering ble utviklet for varehandel og forbrukerrettet virksomhet, og stjerneskjemaer vil nesten alltid fungere meget bra innen slik virksomhet. Hovedårsaken til dette er at dimensjonstabellene stort sett er små i forhold til faktatabellene. I andre typer virksomhet er forholdet mellom fakta- og dimensjonstabeller ofte annerledes, og det er da ikke sikkert "rene" stjerneskjemaer er like effektive. Kimball (2013) har løsninger på slike problemstillinger, som blir tatt opp i neste kapittel. Poe, Klauer og Brobst mener man i stedet kan opptre mye mer fritt i forhold til modelleringen, ut fra hva som er formålstjenlig i den enkelte bransje. Et reisebyrå vil for eksempel arbeide under helt andre forhold og ha helt andre krav enn varehandel.

9.3 Snøflakskjemaer

I et stjerneskjema er dimensjonene denormaliserte. En variant av stjerneskjemaet får vi hvis vi normaliserer dimensjonene. Denne varianten kalles et snøflakskjema, fordi formen kan minne om et snøflak. Et eksempel på et snøflakskjema er vist i figur 9.6.



Figur 9.6 Snøflakskjema (etter Kimball 2013)

Snøflaking frarådes generelt. Kimball (2013) regner opp flere gode grunner til å unngå bruk av snøflakskjemaer. Et argument er at spørringer mot et slikt skjema fort kan bli svært ressurskrevende. Et annet argument er at et slikt skjema er mer komplisert for

brukerne enn et stjerneskjema. Videre vil snøflaking spolere muligheten for å bruke bitmap-indekser. Slike indekser gjør det mulig å indeksere felter med lav kardinalitet. Ved å bruke bitmap-indekser i vanlige stjerneskjemaer kan man i stedet få til en slags ”intern snøflaking”, mens modellen fortsatt er et stjerneskjema. Oracle Warehouse Builder gjør bruk av denne teknikken ved å muliggjøre definisjon av hierarkier.

9.4 Granularitet

Detaljnivået på data i faktatabellen kalles granularitet (”kornethet”). Granularitet er et relativt begrep, idet vi kan si at en tabell har større eller mindre granularitet. Når data hentes fra et transaksjonsprosesseringssystem vil den høyeste granulariteten være enkelttransaksjoner. I dag er det vanlig at man har transaksjoner som granularitet, noe som ikke var like vanlig for noen år siden. Årsaken til dette er at datavolumene i datavarehuset kan bli enorme med transaksjonsdata. For noen år tilbake ble håndtering av slike datamengder for kostbart, idet det krever store platelagre og kraftige maskiner, fortrinnsvis med parallellprosessering.

I dag er teknologien så avansert og rimelig at det sjelden er noen grunn til ikke å bruke transaksjonsdata. På den annen side: hvis man ikke har bruk for transaksjonsdata, kan det være grunn til å velge en annen granularitet.

De store dagligvarekjedene har enorme mengder transaksjonsdata som skal inn i datavarehusene. Selve transaksjonene skjer i butikkenes kasser. En transaksjon vil for eksempel være kjøp av et kneippbrød, en annen kjøp av tre liter lettmelk. Transaksjoner fra kassene samles i butikkens lokale database for salgsdata, og overføres derfra til kjedens sentrale datavarehus. Hver transaksjon vil være knyttet til ikke bare varen som ble solgt, men også butikken den ble solgt i og kassen.

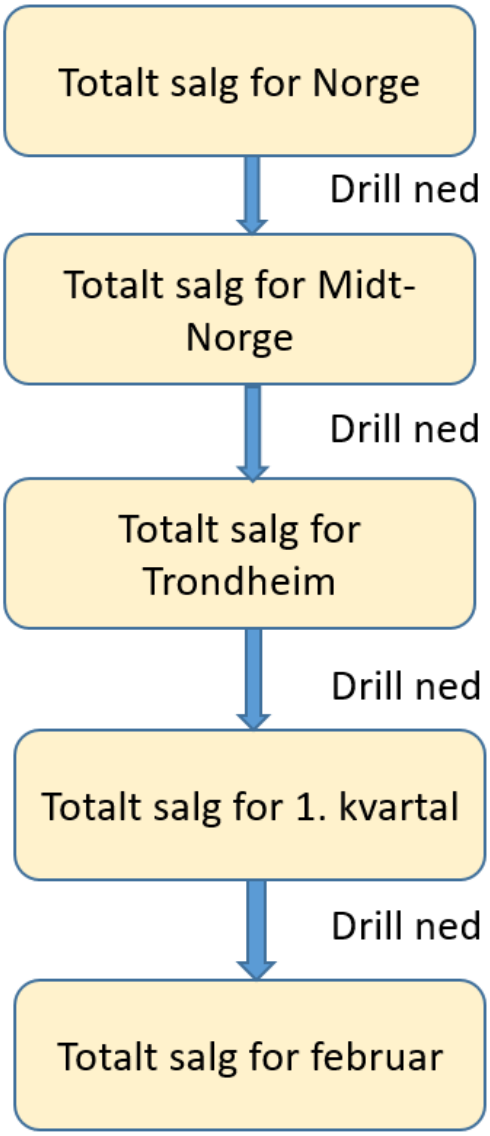
Tidligere var det vanlig å summere salg per vare og butikk per dag i det sentrale datavarehuset. I stedet for flere titalls eller hundretalls transaksjoner med salg av lettmelk i hver enkelt butikk, summerer man altså opp totalt antall liter lettmelk for hver butikk for hver dag. På denne måten reduseres antallet poster i faktatabellen dramatisk.

En slik granularitet setter imidlertid grenser for hva man kan gjøre av analyser på dataene. I dag er en av de viktigste typene analyser på datavarehus såkalt data mining, som går ut på å finne skjulte sammenhenger og mønstre i data. Slik analyse krever data på transaksjonsnivå. Faktatabeller med data som er summert som beskrevet foran vil mangle en del av den informasjonen som finnes i transaksjonsdata.

9.5 Aggregering og drilling

Selv om det kan være viktig å ha mest mulig detaljerte data, er det sjelden det er disse brukerne skal se på. Det vanlige er mer eller mindre summerte tall, slik det er gitt eksempel på i kapittel 10.1.Tallene kan summeres opp på flere nivåer. Drilling er en

teknikk for å navigere fra nivåer med aggregerte data til nivåer med mer detaljerte data. Prinsippet for dette er vist i figur 9.7.



Figur 9.7 Prinsippet bak drilling

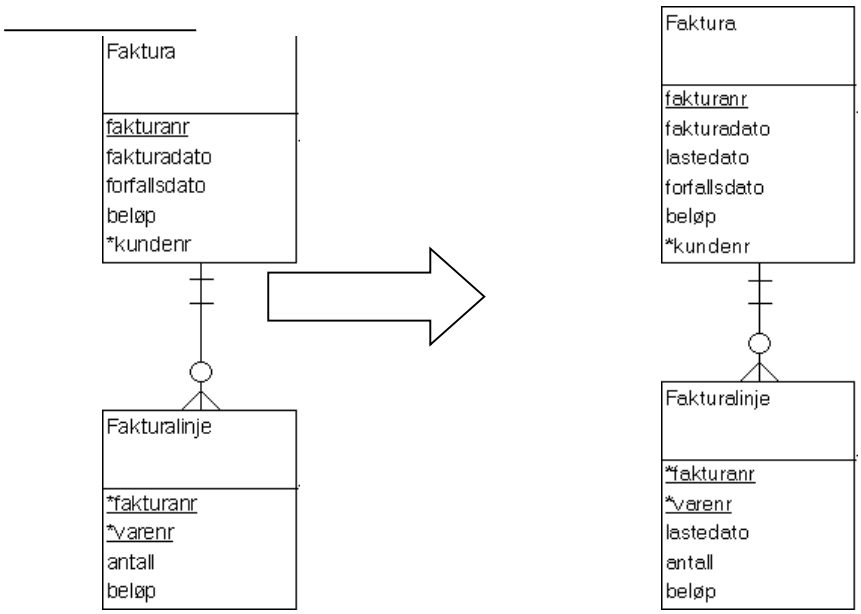
I det viste eksempelet kan vi tenke oss et konsern med anlegg i hele landet. Det øverste nivået med aggregeringer er dermed omsetningstall for hele landet. Flere bedrifter av denne typen har et Norgeskart som startside, der salgsregioner eller fylker vises i forskjellige farger. Grønt viser at alt går etter budsjett, rødt at det ligger under budsjett og gult at man er i faresonen. Dette er management by exception.

Lederen kan nå drille seg til mer detaljerte data. I eksempelet har man valgt ut salgsdistrikt Midt-Norge. La oss si at konsernet har flere anlegg i regionen. Den videre drillingen vil gå ut på å gå inn på en enkelt by og få opp data for denne. Derfra kan man drille enda dypere, til et bestemt kvartal og derfra videre til en enkelt måned.

Systemet må håndtere aggregeringer av data, enten ved at det summerer opp i drillingprosessen eller ved at det bruker ferdig aggregerte data. Bruk av summeringsfunksjonen i SQL på store datamengder er temmelig ressurskrevende. I datavarehussammenheng vil det som regel være et mål å redusere prosesseringstiden for data, rett og slett fordi datavarehuset typisk inneholder så mye av dem. En måte å redusere prosessering på, er å lage ekstra faktatabeller med ferdig summerte data. I stedet for å utføre en SQL-setning med summering, kan sluttbrukerverktøyet bare hente de ønskede data i den aggregerte tabellen. Dette krever imidlertid at sluttbrukerverktøyet "vet om" tabellen, og automatisk redirigerer spørringen dit. For brukeren skal dette være transparent, det vil si foregå bak kulissene uten av brukeren merker noe.

9.6 Modellering av sentralt datavarehus til 3NF

Et sentralt datavarehus etter Inmons modell, gjerne kalt et Enterprise Data Warehouse, modelleres mer eller mindre etter klassiske prisnipper. Det betyr at utviklerne her normaliserer modellen til 3NF eller BCNF.



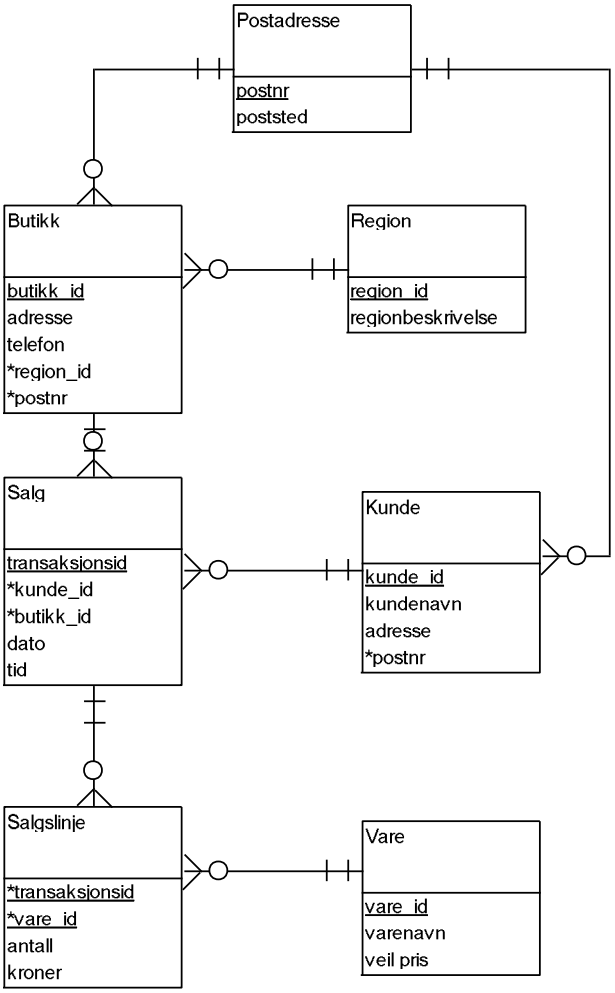
Figur 9.8 Normalisering i kildesystemet (til venstre) og datavarehuset (til høyre)

Datavarehusets datamodell skiller seg fra kildesystemenes ved at bare relevante data er tatt med. Dette gjelder både tabeller og attributter. For de data som lastes inn i datavarehuset legges et "tidsstempel" (time stamp) til primærnøkkelen. Dette tidsstempelet er gjerne tidspunkt for lasting av data (Lindsted).

Datamodellen vil på den annen side også være mer omfattende enn kildesystemenes. Dette skyldes at en virksomhetsmodell skal kombinere datamodeller fra flere kildesystemer (Sperley 1999). Et sentralt problem med å modellere sentrale datavarehus i store organisasjoner, er at man arbeider mot kildesystemer som er i stadig forandring.

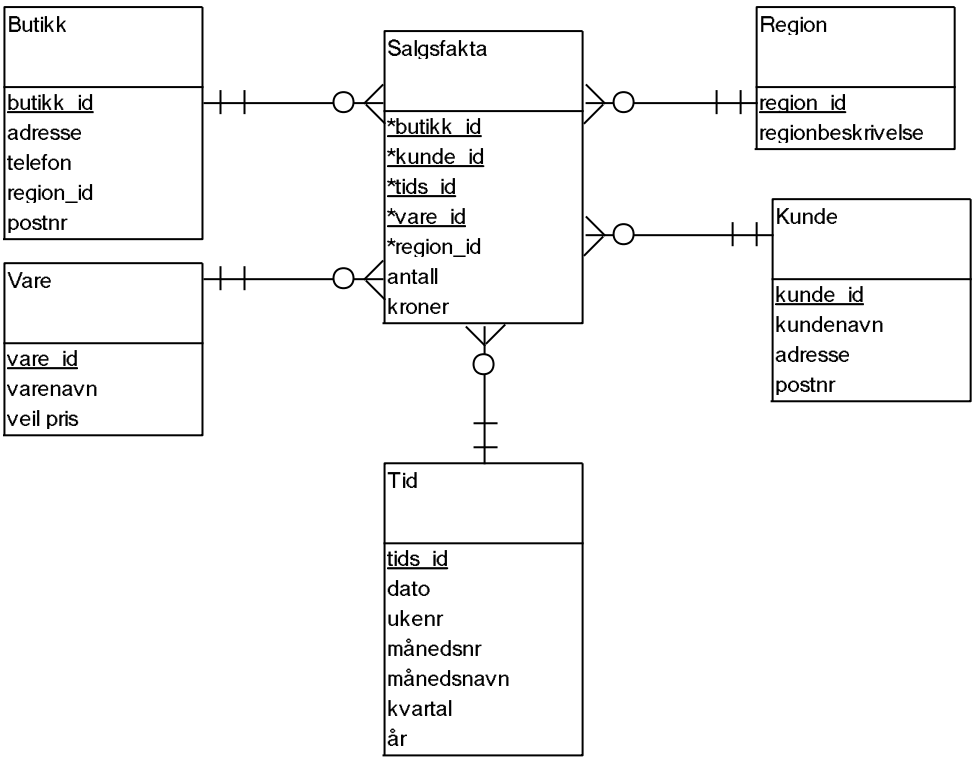
Noen systemer videreutvikles, nye kommer til og andre forsvinner. Forsøk på å lage en sentral datavarehusmodell kan dermed bli evigvarende. Konsekvensen av dette er at man ikke må ha ambisjoner om å få ferdig en komplett modell før man lager selve datavarehuset.

Sperley anbefaler å modellere akkurat nok til at man kan komme i gang. Prinsippet er dermed at også et normalisert datavarehus vil vokse etter hvert som nye bruksområder kommer til. Nedenfor er et eksempel på en normalisert datavarehusmodell.



Figur 9.9 Normalisert datavarehusmodell for salg (Sperley 1999)

Modellen gjelder for salg der kundene kan identifiseres (som med bensinselskaper og for eksempel elektriske forretninger). Det sentrale datavarehuset er imidlertid bare en mellomstasjon på veien til dimensjonsbaserte data marts. Av grunner vi tidligere har diskutert, er det disse sluttbrukerne forholder seg til. Modellen i figur 10.9 er grunnlaget for denne dimensjonsmodellen som brukerne blir presentert for:



Figur 9.10 Dimensjonsmodell basert på modellen i figur 10.9

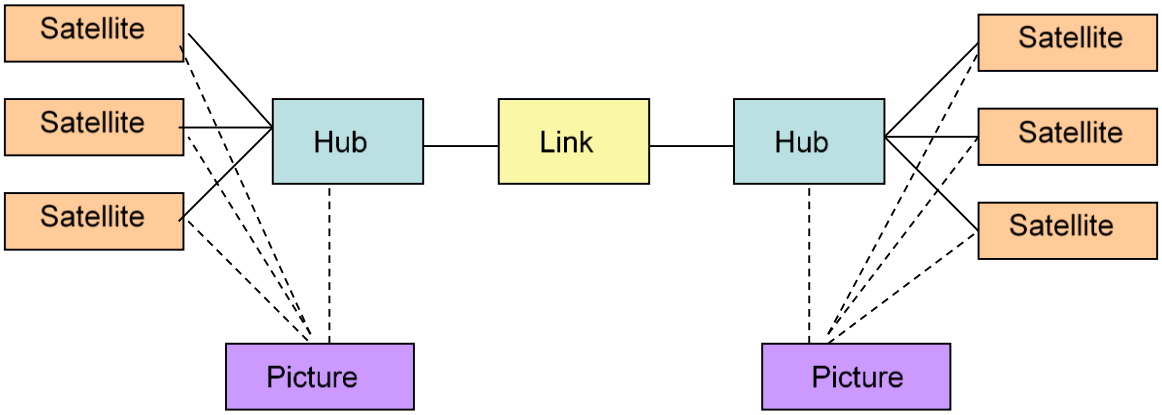
Problemene med normaliserte datavarehus er knyttet til fleksibilitet og skalerbarhet (Lindstedt).

9.7 Modellering av sentralt datavarehus som Data Vault

Data vault er et nytt prinsipp for modellering av sentrale datavarehus. Det er utviklet av Dan Linstedt, og hensikten er å kombinere de sterke sidene ved normaliserte modeller og dimensjonsmodeller. Det sentrale er å gjøre datavarehuset fleksibelt og skalerbart.

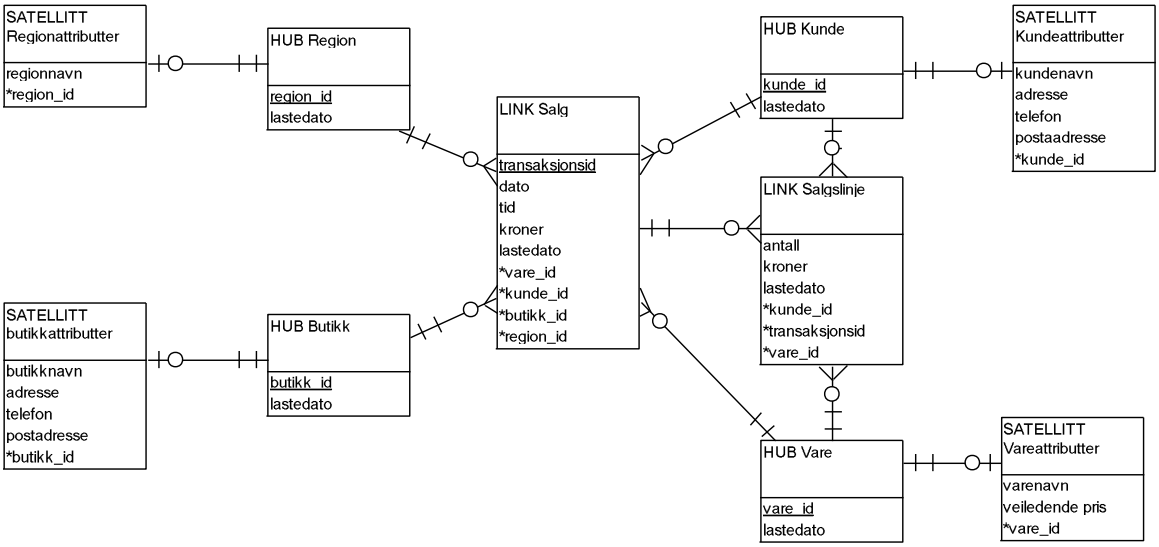
Et Data Vault består av fire forskjellige entitetstyper (Lindstedt):

- Hub entitetstyper (hubs): en hub har som oppgave å koble sammen andre tabeller, og inneholder derfor først og fremst nøkler. En hub vil også inneholde en surrogatnøkkel. Andre attributter kan være tidsstempler og referanse til datakilde. Hub'ene representerer forretningssubjekter, dvs. tilsvarende dimensjoner i en dimensjonsmodell.
- Link entitetstyper (links): Dette er entitetiseringer av m:n-relasjonstyper. En link binder sammen to hub'er. En link representerer også forretningsprosesser, og det er her v vil legge det som i dimensjonsmodellene kalles fakta.
- Satellitt-entitetstyper: Satellittene inneholder beskrivende data for hub'ene, mer presist de attributtene der verdiene kan forandre seg over tid. I et data vault er altså satellittenes oppgave å isolere de foranderlige attributtene fra den øvrige.
- Picture: Disse kalles også tidspunkttabeller. Angir hvilke attributtverdier som er gyldige på et gitt tidspunkt.



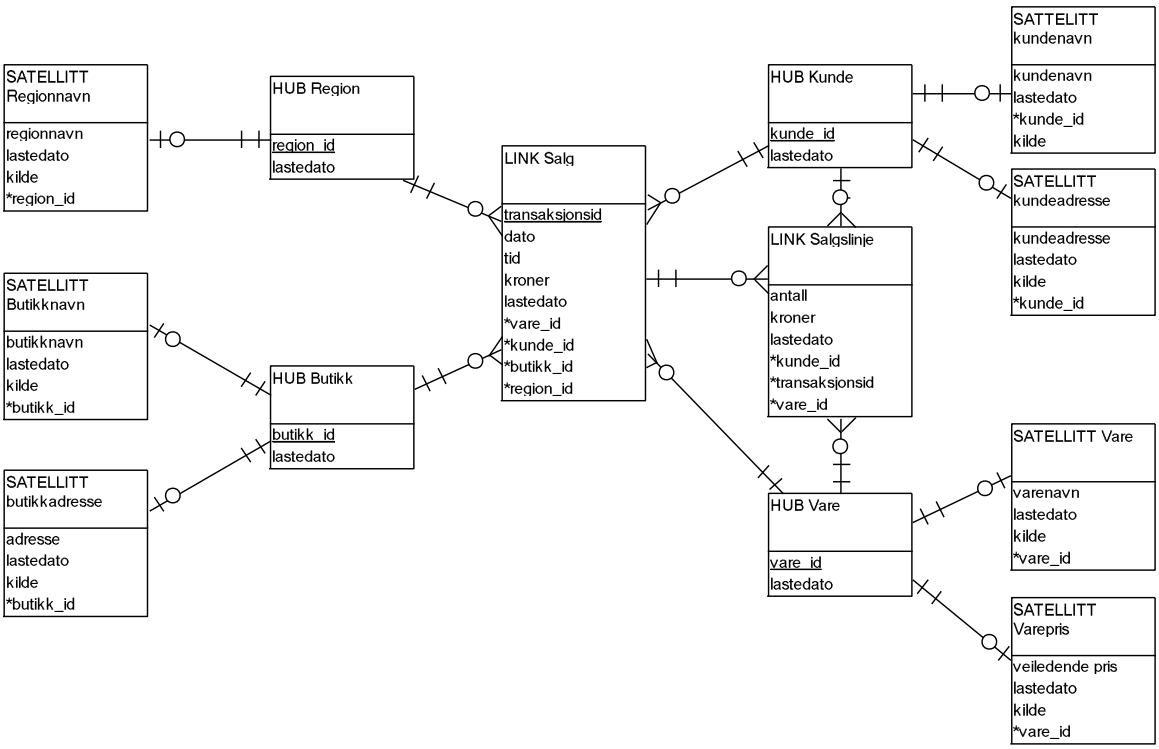
Figur 9.11 Data Vault-modell (etter Lindstedt)

Vi skal nå presentere en Data Vault-utgave av den normaliserte datamodellen fra figur 10.10. Vi ser her bort fra tidspunkttabellene.



Figur 9.12 Data Vault-modell for salg

Modellen kan splittes opp ytterligere, ved at attributtene i satellittabellene legges i hver sin tabell sammen med lastedato for attributtet.



Figur 9.13 Data Vault-modell med en satellitt for hvert attributt

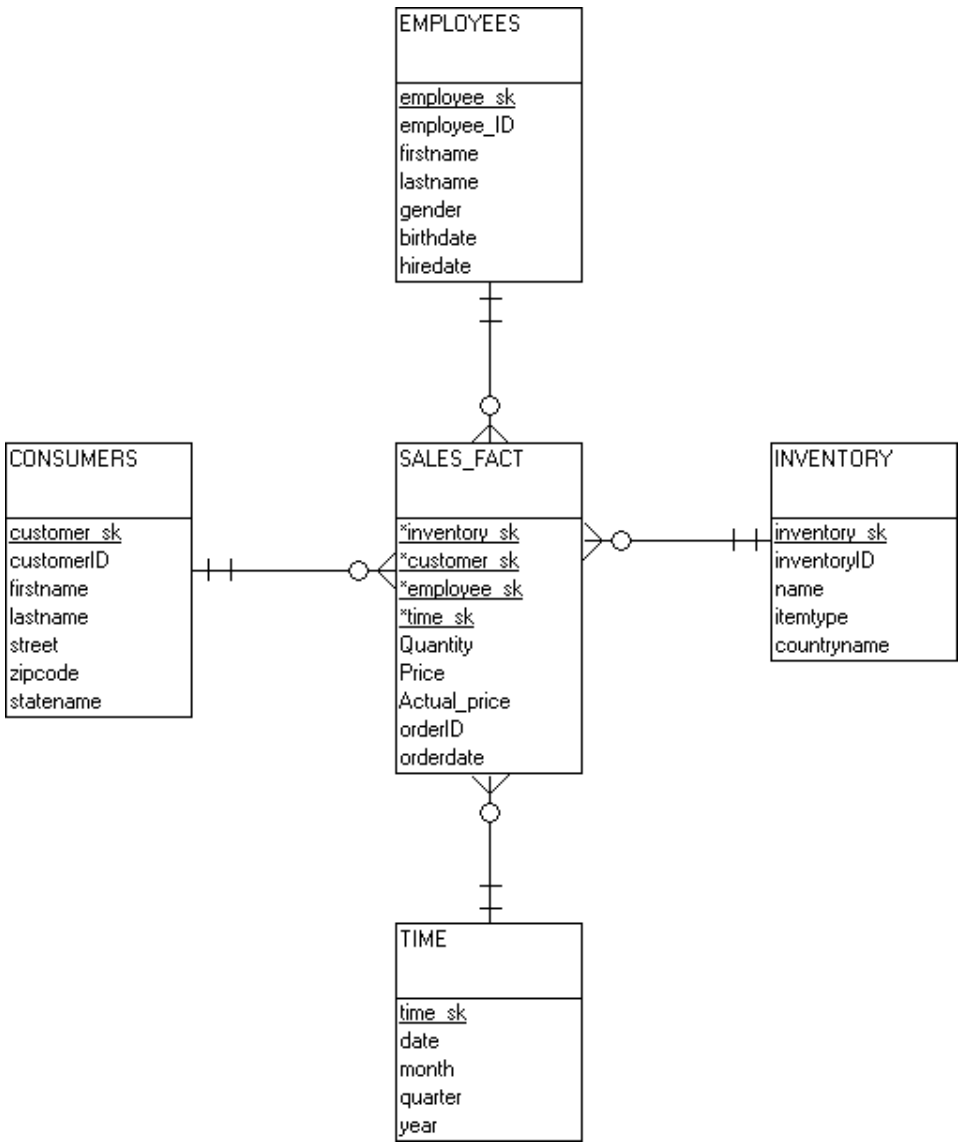
En ulempe med Data Vault-modellen er at den er vanskelig å forstå. Selv om den bygger på et standard rammeverk, er den heller ikke egnet for sluttbrukerverktøy på samme måte som dimensjonsmodeller. Men, som med andre normaliserte modeller, skal også et Data Vault-datavarehus være et grunnlag for dimensjonsmodellerte data marts.

En annen ulempe er at den sterke oppsplittingen av tabeller krever mer fysisk lagringsplass enn andre modeller.

9.8 Datavarehus for Coffeemaker

Datamodell for et enkelt datavarehus

Vi skal lage et enkelt datavarehus for CoffeeMerchant basert på stjerneskjema. I utgangspunktet skal vi bare ha én faktatabell med transaksjonsdata. Denne modellen ser slik ut:



9.8.1 Opprette tidstabell

```
USE coffeemerchant_dw;
CREATE TABLE dim_tid (
   dato_sk INT NOT NULL AUTO_INCREMENT PRIMARY KEY,
   dato DATE,
   maanedsnavn CHAR(9),
   maaned INT(2),
   kvartal INT(1),
   aar INT(4));
```

For å fylle opp tidsdimensjoner med datoer, bruker vi dette scriptet:

```
USE coffeemerchant_dw;
DROP PROCEDURE IF EXISTS pre_fyll_tidsdimensjon;
DELIMITER //
CREATE PROCEDURE pre_fyll_tidsdimensjon (IN start_dato DATE,IN
slutt_dato DATE)
BEGIN
  WHILE start_dato < slutt_dato DO
    INSERT INTO dim_tid
    VALUES (
      NULL,
      start_dato,
      MONTHNAME(start_dato),
      MONTH(start_dato),
      QUARTER(start_dato),
      YEAR(start_dato));
      SET start_dato = ADDDATE(start_dato,1);
  END WHILE;
END //
DELIMITER ;//
```

Vi kjører scriptet ved å kalle prosedyren med startdato og sluttdato som parametere. Siden databasen inneholder data fra 2005 og 2006, holder det med disse to årene:

```
CALL pre_fyll_tidsdimensjon('2005-01-01','2006-12-31');
```

Tidsdimensjonen ser nå slik ut:

dato_sk	dato	maanedsnavn	maaned	kvartal	aar
1	2005-01-01	January	1	1	2005
2	2005-01-02	January	1	1	2005
3	2005-01-03	January	1	1	2005
4	2005-01-04	January	1	1	2005
5	2005-01-05	January	1	1	2005
6	2005-01-06	January	1	1	2005
7	2005-01-07	January	1	1	2005
8	2005-01-08	January	1	1	2005
9	2005-01-09	January	1	1	2005
10	2005-01-10	January	1	1	2005
11	2005-01-11	January	1	1	2005
12	2005-01-12	January	1	1	2005
13	2005-01-13	January	1	1	2005
14	2005-01-14	January	1	1	2005

9.8.2 Opprett øvrige dimensjoner

Det er et prinsipp for datavarehus at vi ikke bruker kildesystemets primærnøkler. I dimensjonstabellene har vi derfor definert surrogatnøkler i tillegg til nøklene som hentes fra kildesystemet.

```
USE coffeemerchant_dw;
CREATE TABLE dim_inventory (
      inventory_sk INT NOT NULL AUTO_INCREMENT PRIMARY KEY,
      InventoryID INT,
       Name        VARCHAR(40) NOT NULL,
       Price       FLOAT(6,2),
       itemType    VARCHAR(1),
       countryname VARCHAR(40));


USE coffeemerchant_dw;
CREATE TABLE dim_employee (
      employee_sk INT NOT NULL AUTO_INCREMENT PRIMARY KEY,
      EmployeeID     INT,
      FirstName      VARCHAR(30) NOT NULL,
      LastName       VARCHAR(30) NOT NULL,
      CommissionRate FLOAT(4,4),
      HireDate       DATE,
      BirthDate      DATE,
      Gender         VARCHAR(1));

USE coffeemerchant_dw;
CREATE TABLE dim_customers (
      customer_sk INT NOT NULL AUTO_INCREMENT PRIMARY KEY,
      customerID INT,
      FirstName   VARCHAR(30) NOT NULL,
      LastName    VARCHAR(30) NOT NULL,
      Street      VARCHAR(50),
      Zipcode     VARCHAR(5),
      City        VARCHAR(50),
      State       VARCHAR(2));
```

Vi henter data fra kildesystemets tilsvarende tabeller med disse scriptene:

```
USE coffeemerchant_dw;
INSERT INTO dim_inventory
SELECT
  NULL,
  inventoryID,
  name,
  price,
  itemType,
  countryname
FROM coffeemerchant.inventory,coffeemerchant.countries
WHERE coffeemerchant.inventory.countryID =
coffeemerchant.countries.countryID;


USE coffeemerchant_dw;
INSERT INTO dim_employee
SELECT
  NULL,
```

```
  employeeID,
  firstname,
  lastname,
  commissionRate,
  hireDate,
  birthDate,
  gender
FROM coffeemerchant.employees;


USE coffeemerchant_dw;
INSERT INTO dim_customers
SELECT
  NULL,
  consumerID,
  firstname,
  lastname,
  street,
  zipcode,
  city,
  state
FROM coffeemerchant.consumers;
```

9.8.3 Lage og fylle faktatabeller

Vi skal lage en faktatabell som inneholder det som er målbart, i dette tilfellet antall og beløp. Vi skal også bruke en staging-tabell (selv om det ikke er nødvendig i dette tilfellet). Det betyr at vi først henter ut data fra tabellene Order og OrderLine i kildedatabasen, og legger dem i staging-tabellen. I denne prosessen ligger også at vi regner ut prisen varen faktisk ble solgt for (ut fra rabattprosenten discount) og totalt salg for varelinjen. Disse attributtene heter henholdsvis actual_price og actual_sold i stagingtabellen.

Vi ser først på scriptene som lager tabellene:

```
USE coffeemerchant_dw;

DROP TABLE IF EXISTS fact_stage_order;

CREATE TABLE  fact_stage_order(
  orderID INTEGER,
  orderdate DATE,
  date_sk INTEGER,
  consumerID INTEGER,
  consumer_sk INTEGER,
  employeeID INTEGER,
  employee_sk INTEGER,
  inventoryID INTEGER,
  inventory_sk INTEGER,
  quantity INTEGER,
  price FLOAT(6,2),
  actual_price FLOAT(10,4),
  actual_sold FLOAT (10,2));

USE coffeemerchant_dw;
```

```
DROP TABLE IF EXISTS fact_sales;
CREATE TABLE fact_sales (
      Fact_pk INT AUTOINCREMENT PRIMARY KEY,
      time_sk INT NOT NULL,
      inventory_sk INT NOT NULL,
      customer_sk INT NOT NULL,
      employee_sk INT NOT NULL,
      orderno INT,
      quantity INT,
      price FLOAT,
      actual_price FLOAT(10,2),
      actual_sold FLOAT(10,2));
```

Deretter kan vi kjøre scriptet som fyller opp stagingtabellen:

```
USE coffeemerchant_dw;
TRUNCATE fact_stage_order;
INSERT INTO fact_stage_order
SELECT
    o.orderID,
    o.orderdate,
    NULL,
    o.consumerID,
    NULL,
    o.employeeID,
    NULL,
    ol.inventoryID,
    NULL,
    ol.quantity,
    ol.price,
    ol.price - ol.price*ol.discount,
    (ol.price - ol.price*ol.discount)*ol.quantity
FROM coffeemerchant.orders o, coffeemerchant.orderlines ol
WHERE o.orderID = ol.orderID;
```

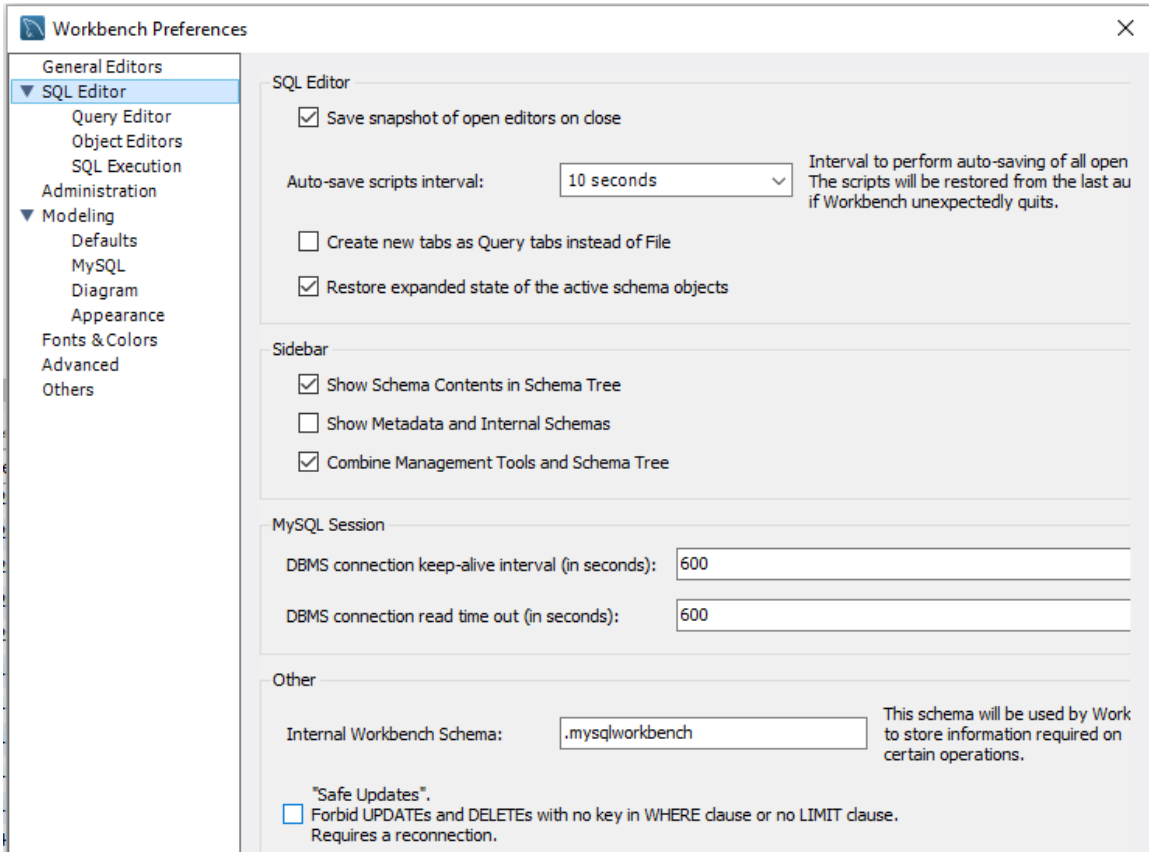
Stagingtabellen ser nå slik ut:

orderID	orderdate	date_sk	consumerID	consumer_sk	employeeID	employee_sk	inventoryID	inventory_sk	quantity	price	actual_price	actual_sold
214010	2005-10-01	NULL	35222	NULL	4058	NULL	236	NULL	18	6.90	6.5550	117.99
214010	2005-10-01	NULL	35222	NULL	4058	NULL	119	NULL	17	13.30	11.9700	203.49
214010	2005-10-01	NULL	35222	NULL	4058	NULL	188	NULL	17	3.90	3.9000	66.30
214010	2005-10-01	NULL	35222	NULL	4058	NULL	122	NULL	14	6.20	5.2700	73.78
214010	2005-10-01	NULL	35222	NULL	4058	NULL	131	NULL	17	10.90	10.3550	176.03
214011	2005-10-01	NULL	33776	NULL	3458	NULL	455	NULL	19	5.30	5.0350	95.67
214011	2005-10-01	NULL	33776	NULL	3458	NULL	398	NULL	8	7.90	7.9000	63.20
214011	2005-10-01	NULL	33776	NULL	3458	NULL	392	NULL	15	7.00	5.9500	89.25
214011	2005-10-01	NULL	33776	NULL	3458	NULL	260	NULL	2	7.10	7.1000	14.20
214011	2005-10-01	NULL	33776	NULL	3458	NULL	458	NULL	10	14.70	12.4950	124.95
214012	2005-10-01	NULL	33271	NULL	1695	NULL	407	NULL	17	4.50	4.5000	76.50
214012	2005-10-01	NULL	33271	NULL	1695	NULL	362	NULL	5	8.40	8.4000	42.00
214012	2005-10-01	NULL	33271	NULL	1695	NULL	452	NULL	14	5.30	4.5050	63.07
214012	2005-10-01	NULL	33271	NULL	1695	NULL	359	NULL	12	8.10	6.8850	82.62
214012	2005-10-01	NULL	33271	NULL	1695	NULL	299	NULL	2	11.90	11.9000	23.80

Her er foreløpig ingen surrogatnøkkel for tid, og for de øvrige dimensjonene er det den opprinnelige primærnøkkelen som vises.

Når vi så leser data over i faktatabellen, skal kildesystemets primærnøkler byttes ut med surrogatnøkler. For å vise hva som skjer, kan vi først illustrere det med fire enkelt-script.

Før vi kjører dem, må vi imidlertid stille om en preferanse. Gå inn på Edit -> Preferences -> SQL Editor, og fjern krysset foran «Safe updates»:



Nå kan vi kjøre scriptene:

```
USE coffeemerchant_dw;
UPDATE fact_stage_order,dim_tid
SET fact_stage_order.dato_sk = dim_tid.dato_sk
WHERE fact_stage_order.orderdate = dim_tid.dato;


USE coffeemerchant_dw;
UPDATE fact_stage_order,dim_employee
SET fact_stage_order.employeeID = dim_employee.employee_sk
WHERE fact_stage_order.employeeID = dim_employee.employeeID;
```

OBS! Ikke kjør dette scriptet! Forklaring neste side.

```
USE coffeemerchant_dw;
UPDATE fact_stage_order,dim_inventory
SET fact_stage_order.inventoryID = dim_inventory.inventory_sk
WHERE fact_stage_order.inventoryID = dim_inventory.inventoryID;


USE coffeemerchant_dw;
UPDATE fact_stage_order,dim_customers
```

```
SET fact_stage_order.consumerID = dim_customers.customer_sk
WHERE fact_stage_order.consumerID =
dim_customers.customerID;
```

Etter dette ser tabellen fact_stage_order slik ut:

orderID	orderdate	date_sk	consumerID	consumer_sk	employeeID	employee_sk	inventoryID	inventory_sk	quantity	price	actual_price	actual_sold
214010	2005-10-01	274	35222	1501	4058	20	236	45	18	6.90	6.5550	117.99
214010	2005-10-01	274	35222	1501	4058	20	119	8	17	13.30	11.9700	203.49
214010	2005-10-01	274	35222	1501	4058	20	188	31	17	3.90	3.9000	66.30
214010	2005-10-01	274	35222	1501	4058	20	122	9	14	6.20	5.2700	73.78
214010	2005-10-01	274	35222	1501	4058	20	131	12	17	10.90	10.3550	176.03
214011	2005-10-01	274	33776	1074	3458	11	455	NULL	19	5.30	5.0350	95.67
214011	2005-10-01	274	33776	1074	3458	11	398	NULL	8	7.90	7.9000	63.20
214011	2005-10-01	274	33776	1074	3458	11	392	NULL	15	7.00	5.9500	89.25
214011	2005-10-01	274	33776	1074	3458	11	260	NULL	2	7.10	7.1000	14.20
214011	2005-10-01	274	33776	1074	3458	11	458	84	10	14.70	12.4950	124.95
214012	2005-10-01	274	33271	928	1695	4	407	NULL	17	4.50	4.5000	76.50
214012	2005-10-01	274	33271	928	1695	4	362	75	5	8.40	8.4000	42.00
214012	2005-10-01	274	33271	928	1695	4	452	NULL	14	5.30	4.5050	63.07
214012	2005-10-01	274	33271	928	1695	4	359	74	12	8.10	6.8850	82.62
214012	2005-10-01	274	33271	928	1695	4	299	56	2	11.90	11.9000	23.80

Legg merke til at det er noen manglende nøkler for inventory. Dette er fordi en del av varenummerene ikke har noe opprinnelsesland fylt inn. Vi må derfor gjøre om scriptet for å fylle varedimensjonen slik at den bruker en OUTER JOIN:

```
USE coffeemerchant_dw;
TRUNCATE TABLE dim_inventory;
INSERT INTO dim_inventory
SELECT
  NULL,
  inventoryID,
  name,
  price,
  itemType,
  countryname
FROM coffeemerchant.inventory
LEFT JOIN coffeemerchant.countries
ON coffeemerchant.inventory.countryID =
coffeemerchant.countries.countryID;
```

Når vi nå kjører oppdatering av surrogatnøkkel i stagingtabellen, blir alt riktig:

orderID	orderdate	date_sk	consumerID	consumer_sk	employeeID	employee_sk	inventoryID	inventory_sk	quantity	price	actual_price	actual_sold
214010	2005-10-01	274	35222	1501	4058	20	236	49	18	6.90	6.5550	117.99
214010	2005-10-01	274	35222	1501	4058	20	119	8	17	13.30	11.9700	203.49
214010	2005-10-01	274	35222	1501	4058	20	188	32	17	3.90	3.9000	66.30
214010	2005-10-01	274	35222	1501	4058	20	122	9	14	6.20	5.2700	73.78
214010	2005-10-01	274	35222	1501	4058	20	131	12	17	10.90	10.3550	176.03
214011	2005-10-01	274	33776	1074	3458	11	455	123	19	5.30	5.0350	95.67
214011	2005-10-01	274	33776	1074	3458	11	398	104	8	7.90	7.9000	63.20
214011	2005-10-01	274	33776	1074	3458	11	392	102	15	7.00	5.9500	89.25
214011	2005-10-01	274	33776	1074	3458	11	260	57	2	7.10	7.1000	14.20
214011	2005-10-01	274	33776	1074	3458	11	458	124	10	14.70	12.4950	124.95
214012	2005-10-01	274	33271	928	1695	4	407	107	17	4.50	4.5000	76.50
214012	2005-10-01	274	33271	928	1695	4	362	91	5	8.40	8.4000	42.00
214012	2005-10-01	274	33271	928	1695	4	452	122	14	5.30	4.5050	63.07
214012	2005-10-01	274	33271	928	1695	4	359	90	12	8.10	6.8850	82.62
214012	2005-10-01	274	33271	928	1695	4	299	70	2	11.90	11.9000	23.80

Endelig kan vi nå laste dataene inn i faktatabellen:

```
USE coffeemerchant_dw;
TRUNCATE TABLE fact_sales;
INSERT INTO fact_sales
SELECT
      NULL,
      orderdate,
     date_sk,
     consumerID,
     consumer_sk,
     employeeID,
     employee_sk,
     inventoryID,
     inventory_sk,
     quantity,
     price,
     actual_price,
     actual_sold
FROM fact_stage_order;
```

Resultatet blir slik:

fact_pk	date	time_sk	consumerID	consumer_sk	employeeID	employee_sk	inventoryID	inventory_sk	quantity	price	actual_price	actual_sold
1	2005-10-01	274	35222	1501	4058	20	236	49	18	6.9	6.55	117.99
2	2005-10-01	274	35222	1501	4058	20	119	8	17	13.3	11.97	203.49
3	2005-10-01	274	35222	1501	4058	20	188	32	17	3.9	3.90	66.30
4	2005-10-01	274	35222	1501	4058	20	122	9	14	6.2	5.27	73.78
5	2005-10-01	274	35222	1501	4058	20	131	12	17	10.9	10.35	176.03
6	2005-10-01	274	33776	1074	3458	11	455	123	19	5.3	5.03	95.67
7	2005-10-01	274	33776	1074	3458	11	398	104	8	7.9	7.90	63.20
8	2005-10-01	274	33776	1074	3458	11	392	102	15	7	5.95	89.25
9	2005-10-01	274	33776	1074	3458	11	260	57	2	7.1	7.10	14.20
10	2005-10-01	274	33776	1074	3458	11	458	124	10	14.7	12.49	124.95
11	2005-10-01	274	33271	928	1695	4	407	107	17	4.5	4.50	76.50
12	2005-10-01	274	33271	928	1695	4	362	91	5	8.4	8.40	42.00
13	2005-10-01	274	33271	928	1695	4	452	122	14	5.3	4.51	63.07
14	2005-10-01	274	33271	928	1695	4	359	90	12	8.1	6.89	82.62
15	2005-10-01	274	33271	928	1695	4	299	70	2	11.9	11.90	23.80

9.9 Flate filer som kilde

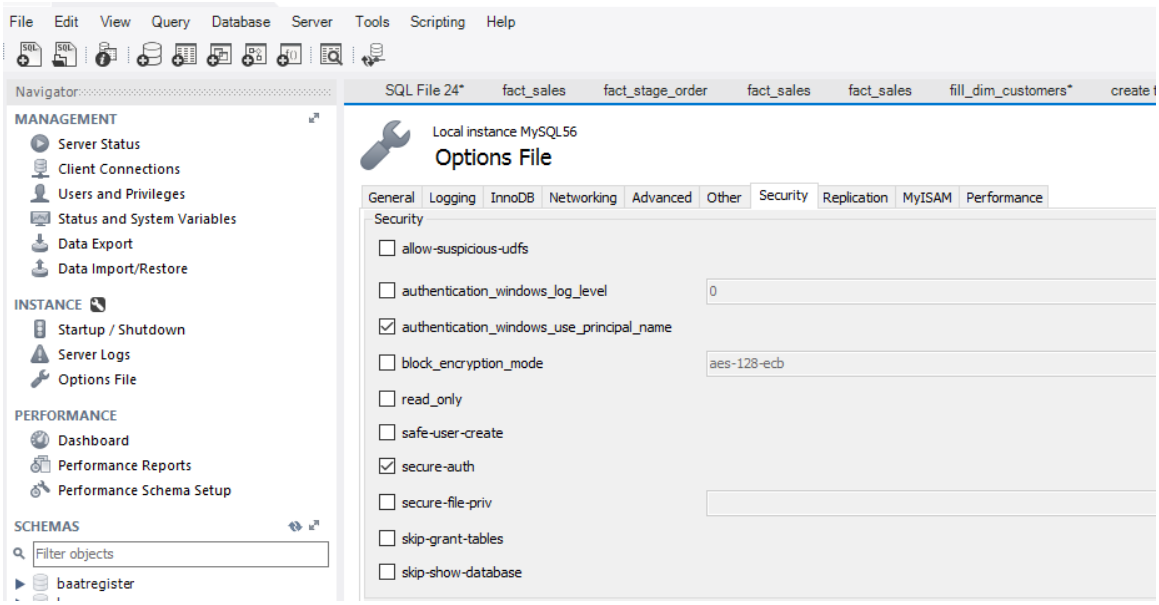
Ofte ligger ikke kildedata på samme type databasesystem som datavarehuset, eller den fysiske plasseringen er slik at vi ikke uten videre kan hente ut data med SQL. Vi kan da i stedet eksportere fra kildesystemet som flate filer, der de enkelte felt er separert med et eller annet tegn. En standard for slike filer er kommaseparerte filer, såkalte csv-filer. Vi skal s på hvordan vi kan eksportere og importere slike filer.

9.9.1 Eksport fra relasjonsdatabase til csv-fil

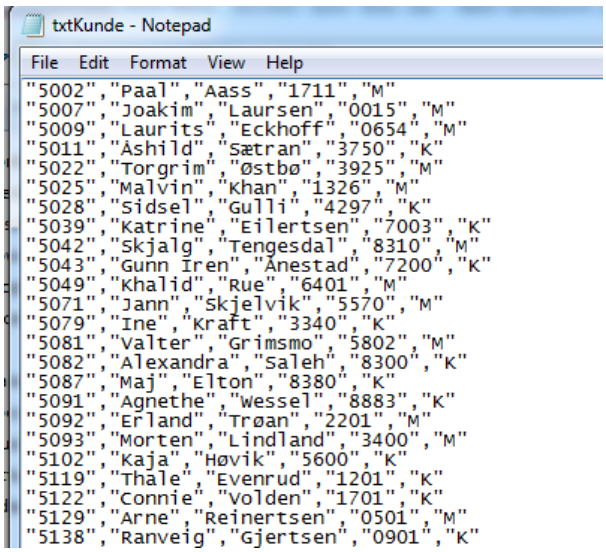
Nedenfor er vist et script som eksporterer fra tabellen Kunde i databasen Hobbyhuset (som hører til læreboken Databasesystemer av Bjørn Kristoffersen):

```
USE hobbyhuset;
SELECT KNr,Fornavn,Etternavn,PostNr,Kjønn
FROM kunde
INTO OUTFILE 'C:/MySql/txtKunde.txt'
FIELDS ENCLOSED BY '"' TERMINATED BY ',' ESCAPED BY '"'
LINES TERMINATED BY '\r\n';
```

Før dere kjører scriptet, må dere gå inn på Options i MySql og passe på at det ikke er haket av for secure-file-priv :



Etter kjøring vil filen txtKunde.txt se slik ut:



9.9.2 Import fra kommaseparert fil til relasjonsdatabasen

Vi kan nå lese fra filen inn i en tabell dim_kunde i MySQL ved hjelp av dette scriptet:

```
USE hobbyhuset_dw;
LOAD DATA INFILE 'c:/mysql/txtKunde.txt'
INTO TABLE dim_kunde FIELDS TERMINATED BY ','
ENCLOSED BY '"'
LINES TERMINATED BY '\n';
```

Kapittel 10 Spørring på datavarehuset med SQL

Vi skal i dette kapittelet se nærmere på hvordan vi kan skrive SQL-spørringer mot datavarehuset. SQL-dialekten i MySQL avviker ørlite fra standarden, men dette er så lite at det ikke har noen betydning. Merk at datoformatet for MySQL er forskjellig fra for eksempel Oracles datoformat.

Det er sjelden det er noen mening i å liste ut detaljdata fra datavarehuset. Det normale er å bruke summeringer, noe vi får ved å bruke GROUP BY.

10.1 Summering av ordre

Vi kan få en utlisting av antall ordre og sum ordre per dag med denne enkle spørringen:

```
SELECT dato,SUM(actual_sold) AS Solgt
FROM dim_tid t,
fact_sales f
WHERE t.dato_sk = f.time_sk
GROUP BY dato
ORDER BY dato;
```

Resultatet ser slik ut:

dato	Solgt
2005-10-01	2302.26
2005-10-02	442.70
2005-10-03	589.18
2005-10-04	899.26
2005-10-05	1240.35
2005-10-06	1305.39
2005-10-07	828.76
2005-10-10	825.64
2005-10-11	1384.53
2005-10-12	1776.60
2005-10-14	2202.75
2005-10-15	479.67
2005-10-16	1328.84
2005-10-17	395.07
2005-10-18	935.08
2005-10-19	740.16

Vi kan utvide spørringen med antall ordre for hver dag:

```
SELECT dato,SUM(actual_sold) AS Solgt,COUNT(*) AS Antall_ordre
FROM dim_tid t,
fact_sales f
WHERE t.dato_sk = f.time_sk
GROUP BY dato
ORDER BY dato;
```

Resultatet ser nå slik ut:

dato	Solgt	Antall_ordre
2005-10-01	2302.26	25
2005-10-02	442.70	11
2005-10-03	589.18	7
2005-10-04	899.26	11
2005-10-05	1240.35	14
2005-10-06	1305.39	13
2005-10-07	828.76	15
2005-10-10	825.64	14
2005-10-11	1384.53	18
2005-10-12	1776.60	22
2005-10-14	2202.75	23
2005-10-15	479.67	5

Disse enkle spørringene gir allerede viktig informasjon til de ansvarlige. Vi skal utvide eksempelet over ved å bore oss nedover til mer detaljerte data. Vi skal legge til fordeling av ordrene på opprinnelsesland:

dato	countryname	Solgt	Antall_ordre
2005-10-01	China	702.44	4
2005-10-01	Ethiopia	140.79	1
2005-10-01	Germany	182.49	2
2005-10-01	Indonesia, Republic of	33.15	1
2005-10-01	Italy	66.30	1
2005-10-01	Japan	42.00	1
2005-10-01	Kenya	255.77	2
2005-10-01	Nicaragua	83.30	2
2005-10-01	Sri Lanka	73.78	1
2005-10-02	China	56.40	2
2005-10-02	Egypt	44.00	1
2005-10-02	Guatemala	7.40	1
2005-10-02	Indonesia, Republic of	132.60	1
2005-10-02	Japan	91.20	1
2005-10-03	Brazil	35.00	1
2005-10-03	China	35.53	1
2005-10-03	Ethiopia	143.73	1
2005-10-03	Guatemala	87.97	1
2005-10-03	India	198.55	2
2005-10-04	Brazil	28.00	1
2005-10-04	China	79.20	1
2005-10-04	Morocco	141.93	1
2005-10-04	Russia	84.15	1
2005-10-04	Tanzania, United Republic of	15.90	1

SQL-spørringen ser slik ut:

```
SELECT dato,countryname,SUM(actual_sold) AS Solgt,COUNT(*) AS
Antall_ordre
FROM dim_tid t,
fact_sales f,
dim_inventory i
WHERE t.dato_sk = f.time_sk
AND i.inventory_sk = f.inventory_sk
GROUP BY dato,countryname
ORDER BY dato,countryname;
```

Dette er et eksempel på hvordan vi utfører drilling med SQL: vi legger til et nytt felt, både i SELECT-delen og under GROUP BY (og nytt tabellnavn dersom det er nødvendig).

10.2 Salg per vare

En typisk rapport for alle virksomheter som selger noe, er rapporter som viser fordeling per vare. Hvis virksomheten har mange forskjellige varer (dagligvarebutikker har fra 3000 – 20000 varenummere, bokhandlere opp til 100000), kan en slik rapport bli helt uoversiktlig. Dette er grunnen til at man opererer med flere nivåer av varegrupper. I vår eksempelbedrift har de bare ett nivå av gruppering: varetype (som er kaffe eller te). En spørring på varetype kan se slik ut:

```
SELECT itemtype As varetype,
SUM(actual_sold) As sum_salg,
COUNT(*) AS antall_solgt
FROM dim_inventory i,
fact_sales_dimensions f
WHERE i.inventory_sk = f.inventory_sk
GROUP BY itemtype
ORDER BY itemtype;
```

Resultatet av denne spørringen ser slik ut:

varetype	sum_salg	antall_solgt
C	42878.46	574
T	73681.73	711

C er kaffe og T er te.

Vi kan nå drille oss ned på varenummer:

```
SELECT itemtype AS varetype,
i.name AS varenavn,
SUM(actual_sold) AS sum_solgt,
COUNT(*) AS antall_solgt
FROM dim_inventory i,
fact_sales_dimensions f
WHERE i.inventory_sk = f.inventory_sk
GROUP BY itemtype,i.name
ORDER BY itemtype,i.name;
```

Resultatet ser vi her:

varetype	varenavn	sum_solgt	antall_solgt
C	Kona Extra Fancy	2127.60	15
C	Kopi Luwak	1247.47	30
C	Mexican Coatepec	1218.40	18
C	Mocha	2860.39	31
C	Mocha Java	1403.52	25
C	New Guinea	1206.94	17
C	Nicaraguan Matagalpa	2274.69	20
C	Sulawesi Peaberry	1401.84	14
C	Sumatra Mandheling	661.08	12
C	Tanzania Moshi	709.07	9
C	Turkish	686.84	13
C	Vienna	595.93	13
C	Yemen Mocha	826.56	13
C	Zimbabwe	1331.46	18
T	An Hui Silver Sprout	1286.97	14
T	Apricot	2650.34	24
T	Assam Fancy 2nd Flush	1727.88	19
T	Assam Tara TGFOP-1	2680.64	19
T	Berry Patch	1753.76	16
T	Ceylon Pekoe Labookelle	558.31	7
T	Ceylon Supreme	1018.15	16
T	Ceylon Uva Highlands	496.10	8
T	Chai	2766.45	20
T	Chamomile Blossom	1558.48	20
T	China Keemun	1739.64	18

En svært nyttig spørring er å få frem hvilke varer man selger mest av. En spørring som sorterer varene i avtagende rekkefølge etter salg i kroner ser slik ut:

```
SELECT itemtype AS varetype,
i.name As varenavn,
SUM(actual_sold) AS sum_solgt,
COUNT(*) AS antall_solgt
FROM dim_inventory i,
fact_sales_dimensions f
WHERE i.inventory_sk = f.inventory_sk
GROUP BY itemtype,i.name
ORDER BY itemtype,sum_solgt DESC;
```

Resultatet av denne spørringen er vist nedenfor.

varetype	varenavn	sum_solgt	antall_solgt
C	Mocha	2860.39	31
C	Jamaican Blue Mountain	2355.10	14
C	Nicaraguan Matagalpa	2274.69	20
C	Celebes Kalossi	2191.28	17
C	Kona Extra Fancy	2127.60	15
C	Kenya AA	1935.02	25
C	Costa Rica La Manita	1700.00	18
C	Brazil Bourbon Santos	1644.67	35
C	Chanchamayo	1478.34	15
C	Columbia Bucaramanga Especial	1461.87	21
C	Ethiopia Harrar	1439.20	17
C	Mocha Java	1403.52	25
C	Sulawesi Peaberry	1401.84	14
C	Zimbabwe	1331.46	18
C	Kopi Luwak	1247.47	30
C	Mexican Coatepec	1218.40	18
C	New Guinea	1206.94	17
C	Brazil Sul De Minas Cerra	1167.72	16
C	Ethiopia Sidamo	1099.41	17
C	Columbia Supremo	1079.96	27
C	Costa Rica Tarrazu	970.14	13
C	Kilimanjaro	968.59	23

10.3 Summering på tid

En annen type rapport er totalsalg per måned. I spørringen nedenfor har vi også tatt med fordeling på år:

```
SELECT aar,
maanedsnavn,
itemtype AS varetype,
SUM(actual_sold) AS sum_salg,
COUNT(*) AS antallsolgt
FROM dim_tid t,
dim_inventory i,
fact_sales_dimensions s
WHERE t.dato_sk = s.time_sk
AND i.inventory_sk = s.inventory_sk
GROUP BY aar,maanedsnavn,itemtype
ORDER BY aar,maanedsnavn,itemtype;
```

Resultatet ser vi her:

år	måned	var...	sum_salg	antallsolgt
2005	December	C	6575.34	90
2005	December	T	13984.41	128
2005	November	C	6848.16	89
2005	November	T	12253.66	112
2005	October	C	6978.56	89
2005	October	T	9687.47	103
2006	April	C	3264.95	50
2006	April	T	6042.49	68
2006	February	C	6331.76	90
2006	February	T	9566.87	93
2006	January	C	6774.89	90
2006	January	T	12473.97	117
2006	March	C	6104.80	76
2006	March	T	9672.86	90

Nå ble ikke dette så bra: månedene kommer hulter til bulter. Men hvis vi i stedet sorterer på månedsnummer, skulle vi få et pent resultat:

```
SELECT aar AS år,
maanedsnavn AS måned,
itemtype AS varetype,
SUM(actual_sold) AS sum_salg,
COUNT(*) AS antallsolgt
FROM dim_tid t,
dim_inventory i,
fact_sales_dimensions s
WHERE t.dato_sk = s.time_sk
AND i.inventory_sk = s.inventory_sk
GROUP BY aar,maanedsnavn,itemtype
ORDER BY aar,maaned,itemtype;
```

Her ser vi resultatet:

år	måned	varetype	sum_salg	antallsolgt
2005	October	C	6978.56	89
2005	October	T	9687.47	103
2005	November	C	6848.16	89
2005	November	T	12253.66	112
2005	December	C	6575.34	90
2005	December	T	13984.41	128
2006	January	C	6774.89	90
2006	January	T	12473.97	117
2006	February	C	6331.76	90
2006	February	T	9566.87	93
2006	March	C	6104.80	76
2006	March	T	9672.86	90
2006	April	C	3264.95	50
2006	April	T	6042.49	68

Kapittel 11 Avansert dimensjonsmodellering

Hittil har vi sett på grunnleggende prinsipper for dimensjonsmodellering. Vi skal i dette kapittelet se på mer avanserte problemstillinger. Vi vil her se en av dimensjonsmodelleringens sterke sider: modellen utgjør et standardisert rammeverk der vi kan behandle forskjellige problemstillinger på en standardisert måte. På den annen side er det forfattere som mener utviklerne bør føle seg frie i forhold til grunnleggende modelltyper som stjerneskjemaer.

Den som først og fremst har behandlet dimensjonsmodellering er Ralph Kimball (Kimball 2013). Det som tas opp i foreliggende kapittel er derfor stort sett basert på hans drøftinger.

11.1 Problemer knyttet til oppdatering av dimensjoner

Mens fakta i faktatabellene i utgangspunktet ikke skal oppdateres, er oppdateringer normalt i dimensjonstabellene. En kunde kan for eksempel flytte eller få nytt telefonnummer, en vare kan få ny pris eller forpakningsstørrelse eller for den saks skyld nytt EAN-nummer. Data i dimensjonstabellen må dermed oppdateres. Spørsmålet er hvordan dette skal gjøres.

Kimball (ibid.) skiller mellom fire forskjellige oppdateringssituasjoner for dimensjoner:

- langsomme endringer i små tabeller
- hyppige endringer i små tabeller
- endringer i store tabeller
- hyppige endringer i svært store tabeller

Hver av disse situasjonene krever spesiell behandling.

11.1.1 Endringer i små tabeller

Med små tabeller mener vi her tabeller med relativt få poster. Det vil selvsagt være et definisjonsspørsmål hva som er få og mange poster, men noen tusen er i denne sammenheng ikke mange. Når vi kommer opp i hundre tusen er vi antagelig over på mange poster.

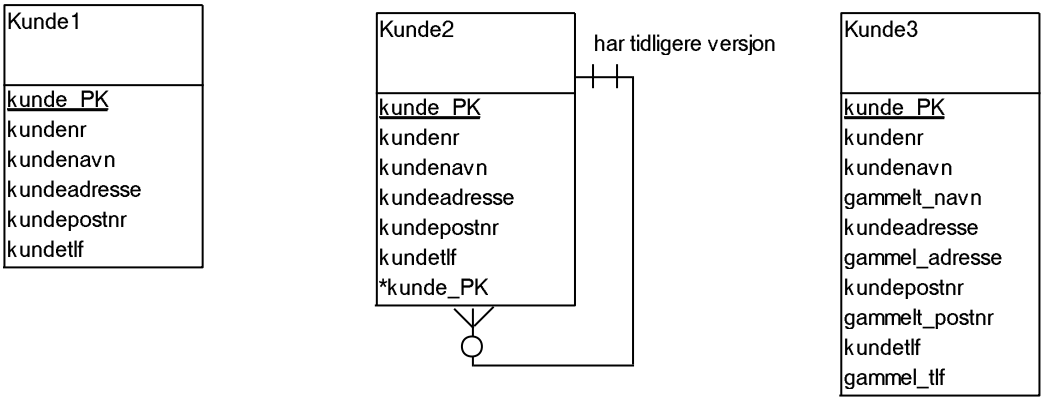
Endringer kan skje både sporadisk og hyppig. Det grunnleggende spørsmålet når det gjelder oppdateringer i dimensjonstabeller er om det er interessant å beholde tidligere verdier. Det er her tre forskjellige strategier vi kan bruke, kalt type 1, type 23 og type 3 behandling av oppdateringer.

Type 1 behandling vil si at vi bare overskriver gammel verdi. Dermed mister vi muligheten for historie.

Type 2 behandling vil si at vi beholder den gamle posten og oppretter en ny post med en ny verdi for surrogatnøkkelen. Den nye posten må ha en referanse til den gamle.

Type 3 behandling vil si at vi alltid beholder den forrige verdien, men ikke mer enn denne. Dette kan gjøre ved at det for hvert felt som kan oppdateres på denne måten, også er et felt med gammel verdi.

De tre strategiene for oppdatering er vist som datamodeller i figur 11.1.



Figur 11.1 Dimensjonen Kunde tilrettelagt for oppdatering av type 1, 2 og 3

Type 1 oppdatering brukes når det ikke er interessant å beholde noen historie. For eksempel vil det sjelden være interessant å kjenne til en kundes forrige telefonnummer. Derimot kan det være interessant å kjenne en kundes forrige postadresse, fordi det kanskje har konsekvenser for omsetningen innen et salgsdistrikt hvis kunden flytter.

Gruppe 2 oppdatering brukes når man ønsker å beholde en fullstendig historie for forekomster i dimensjonen. Et eksempel på en situasjon der dette kan være interessant er prisen på en vare. Selv om det bare er et attributt som er forandret, vil en spørring på dette attributtet gi tilstanden til hele forekomsten på forskjellige tidspunkt, bestemt av den aktuelle tilstanden til attributtet.

Gruppe 3 oppdatering brukes når det kan være interessant å kjenne til at en forandring har skjedd, uten at denne forandringen betyr noe. For eksempel vil det ofte være interessant av praktiske årsaker å kjenne den eventuelle forrige adressen til en kunde, uten at det brukes i spørringer.

Med sporadiske forandringer av attributtverdier kan man velge mellom alle tre strategiene for oppdatering. Med hyppige oppdateringer (daglig eller mange ganger i året) vil det ofte være mest aktuelt å bruke type 2 oppdatering for å kunne spore alle endringer (Kimball 2013).

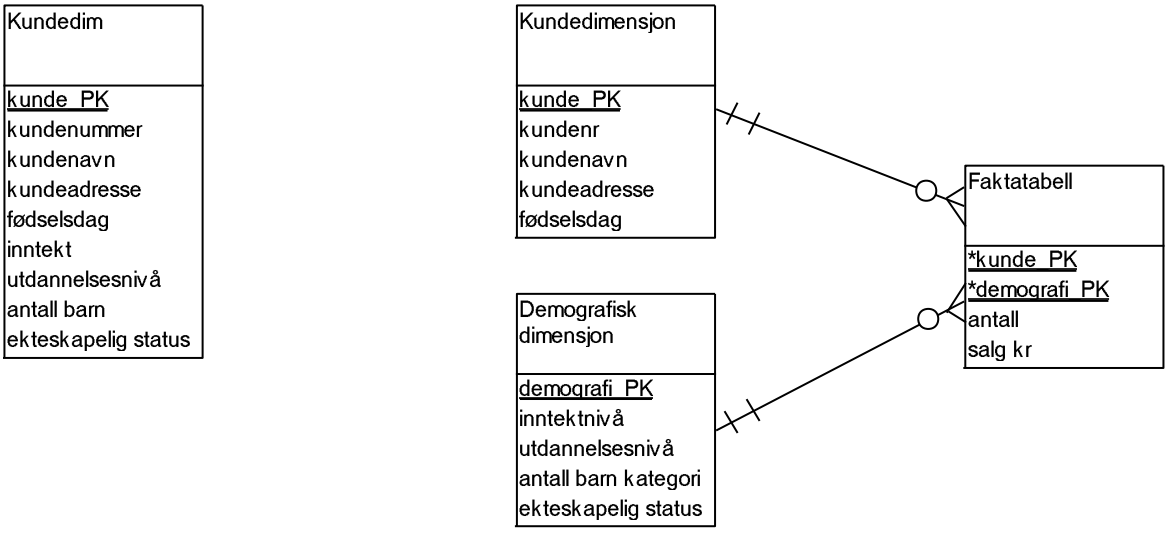
11.1.2 Store dimensjonstabeller

Når dimensjonstabellene blir store, det vi si at de får svært mange poster, anbefales generelt å ikke ta vare på historien ved endringer. Antallet poster er allikevel stort nok. For å ta noen eksempler: en varedimensjon innen bokhandlerbransjen kan typisk omfatte 100 000 artikler. For offentlig virksomhet i Norge vil en brukerdimensjon ha 5 millioner poster, mens de store amerikanske butikkjedene kan ha registrert mer enn 100 millioner

kunder. Med slike størrelser på tabellene vil både strategi 2 og 3 skape mye ekstra data, og det er noe man vanligvis vil unngå.

Dette gjelder spesielt når endringene blir hyppige på store dimensjonstabeller, noe som typisk kan være tilfelle for dimensjoner med persondata. En måte å håndtere slike situasjoner på er å splitte dimensjonsmodellen, slik at de attributtene som kan endres skilles ut i en egen tabell. Det er jo typisk bare noen attributter, som adresse og telefon, som kan endres hos en kunde. Navn og fødselsdato endres normalt ikke.

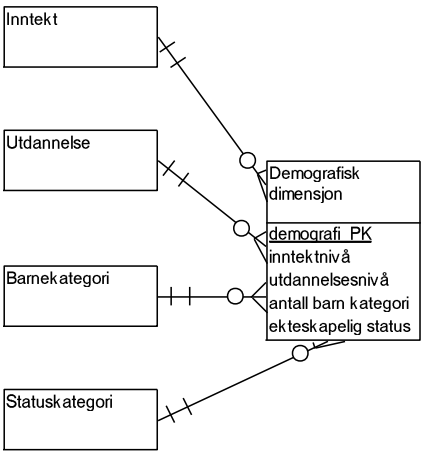
Figur 11.2 viser en datamodell av en splitting av kundedimensjonen. Her er ikke brukt "snøflaking" av kundedimensjonen; i stedet er en demografisk dimensjon lagt til og koblet direkte til faktatabellen.



Figur 11.2 Utskilling av attributter utsatt for endring i egen dimensjon

En viktig del av denne utskillelsen er at vi nå forandrer domenene til de foranderlige attributtene. I stedet for "frie verdier" for eksempel for inntekt, vil vi nå operere med et begrenset antall kategorier. Det samme gjelder for de øvrige attributtene. Ved interessante forandringer i data for en kunde kan vi nå bare forandre dataverdiene i den demografiske tabellen. Denne og selve kundetabellen er knyttet sammen gjennom faktatabellen (det er altså ikke nødvendig med noen fremmednøkkel for kunde i demografidimensjonen).

Egentlig ser den demografiske dimensjonen slik ut, med oppslagstabeller:



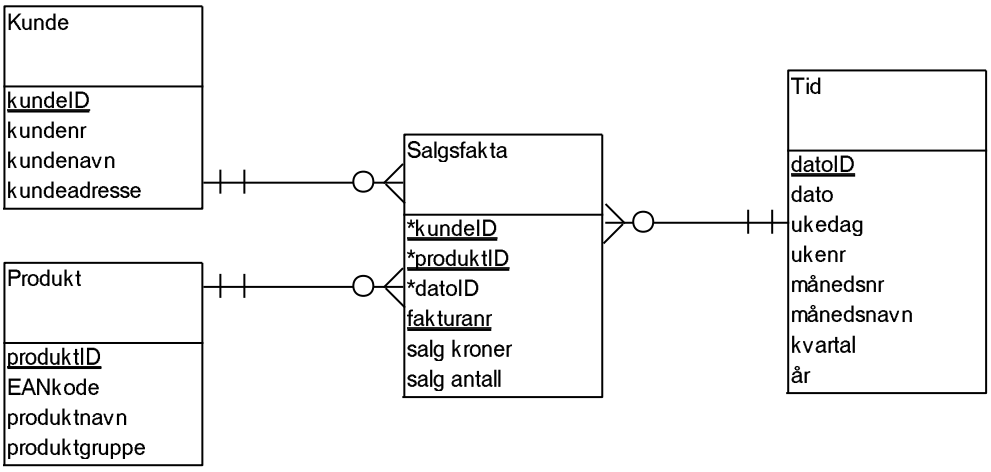
Figur 11.3 Demografisk dimensjon med oppslagstabeller

11.2 Modeller med variasjoner over fakta og dimensjoner

Noen dimensjonsmodeller kan avvike fra normalen enten ved at de har dimensjoner som ikke er representert ved dimensjonstabeller, eller at de har faktatabeller uten fakta. I denne gruppen kan vi også regne faktatabeller der vi ønsker å knytte eksakt tidspunkt til transaksjoner.

11.2.1 Degenererte dimensjoner

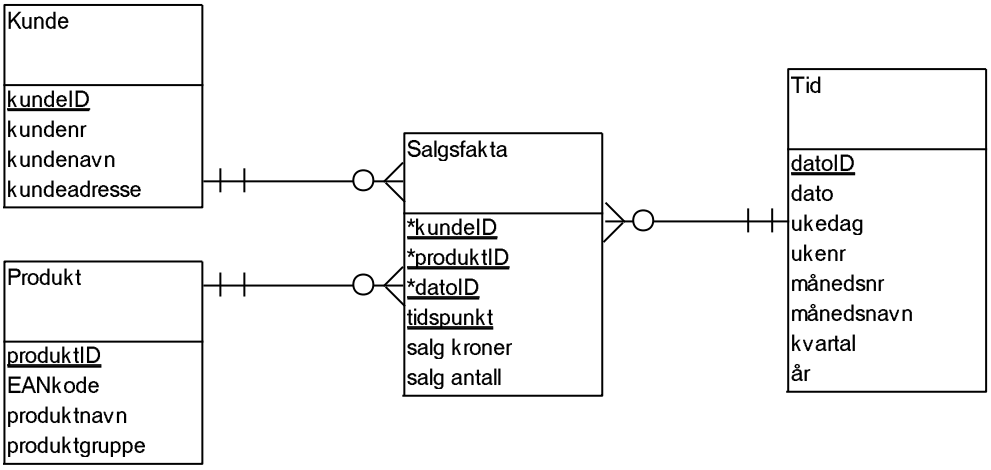
I noen tilfelle kan vi ha dimensjonsdata som egentlig ikke har noen dimensjon. Et typisk eksempel er fakturanummere og ordrenummere. Hvis vi ønsker å ha med disse trenger vi ikke noen egen dimensjon. Nummeret legges inn i faktatabellen på samme måte som fremmednøklene til dimensjonstabellene, men uten at det refererer til noen dimensjon. Dette er det Kimball kaller en degenerert dimensjon.



Figur 11.4 Dimensjonsmodell med degenerert dimensjon

11.2.2 ”Timestamp” på fakta

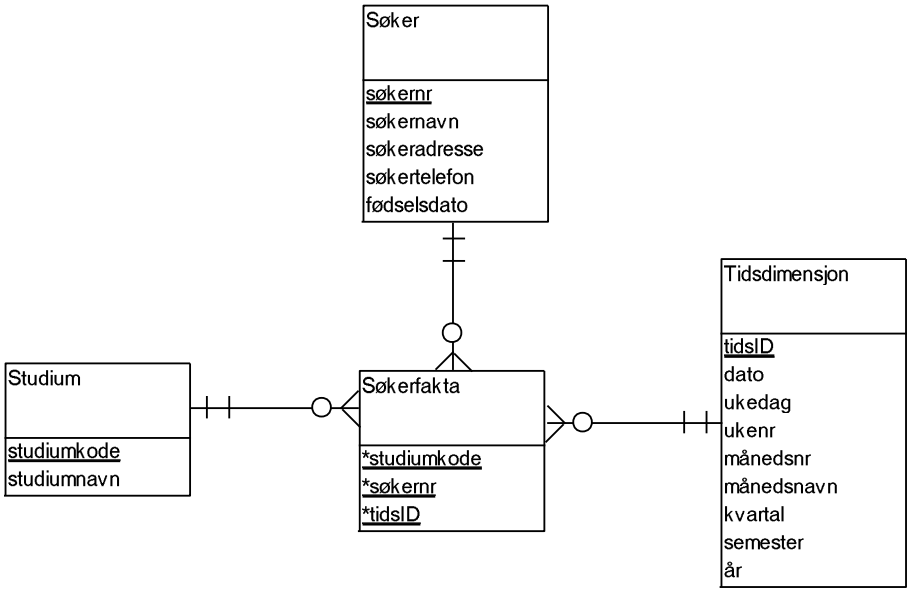
Hvis vi ønsker et ”timestamp” på poster i faktatabellen, gjøres det på samme måte som med degenererte dimensjoner. Fremmednøkkelen for tid i faktatabellen bør typisk referere til en dato. Dersom vi skal ha med klokkeslett, vi en tidsdimensjon basert på slike bli altfor omfattende. Løsningen er å ha en tidsreferanse til tidsdimensjonen, der dato er enhet, og i tillegg et klokkeslett i faktatabellen.



Figur 11.5 Bruk av timestamp i faktatabellen

11.2.3 Faktaløse faktatabeller

En degenerert dimensjon kan beskrives som en ”dimensjon uten dimensjonstabell”. Vi kan imidlertid også ha faktatabeller uten fakta. I disse tilfellene brukes faktatabellen til å registrere hendelser som det ikke er knyttet noen fakta til. En slik hendelse kan for eksempel være at en student har søkt opptak til en høyskole. Det er ikke knyttet noen fakta til denne hendelsen, men den kan knyttes opp til dimensjoner som søker, studium og studieår. Vi kan derfor lage følgende dimensjonsmodell:



Figur 11.6 Dimensjonsmodell med faktaløs faktatabell

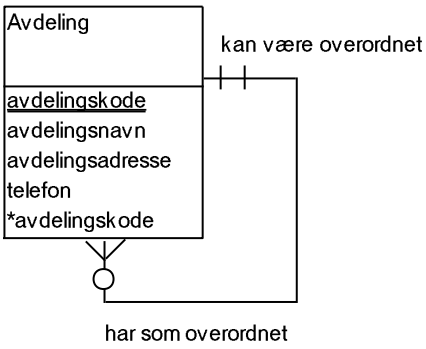
11.3 Dimensjonsmodeller med flere faktatabeller

Det er i praksis aldri sli at et datavarehus bare inneholder en faktatabell. Det typiske er mange forskjellige faktatabeller. Noen av disse er selvstendige i den forstand at de inneholder fakta som ikke har noen sammenheng med fakta i andre faktatabeller. I andre tilfelle vil forskjellige faktatabeller høre sammen på forskjellige måter. Vi skal her se nærmere på slike sammenhenger.

11.3.1 Hierarkier uttrykt gjennom faktatabeller

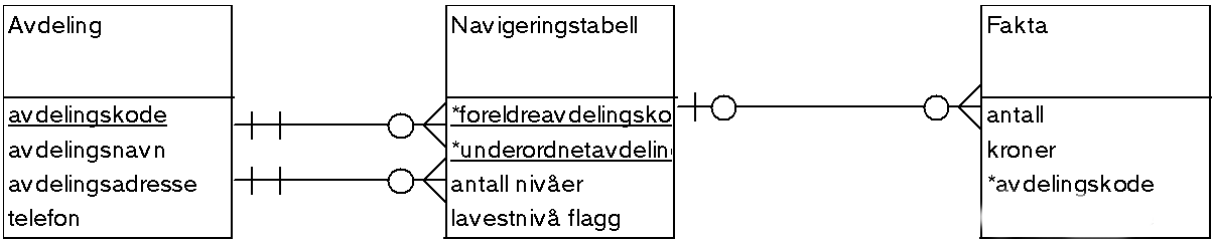
Hierarkier finner vi mange av i den virkelige verden. Typiske hierarkier vi kommer i kontakt med som databaseutviklere er knyttet til organisasjoner og produktdeler. Vi har i tidligere kurs sett på håndtering av dette ved hjelp av egenrelasjoner. Et eksempel på dette er vist nedenfor.

Problemet i datavarehussammenheng er at vi ikke får brukt GROUP BY-utsagn på en slik datamodell. GROUP BY kan ikke følge et hierarki gjennom egenrelasjoner. Oracle har faktisk en operasjon som kan gjøre dette, nemlig CONNECT BY, men denne kan ikke brukes sammen med en JOIN. Vi trenger derfor en egen datamodell som gjør det mulig å bruke GROUP BY (noe som er helt nødvendig for at vi skal kunne gjøre fornuftige spørringer).



Figur 11.7 Typisk selvrefererende hierarki

I datavarehuset kan vi løse dette problemet ved en form for entitetisering, selv om vi her har en 1 : n relasjonstype.



Figur 11.8 Navigeringsbro for hierarkisk struktur

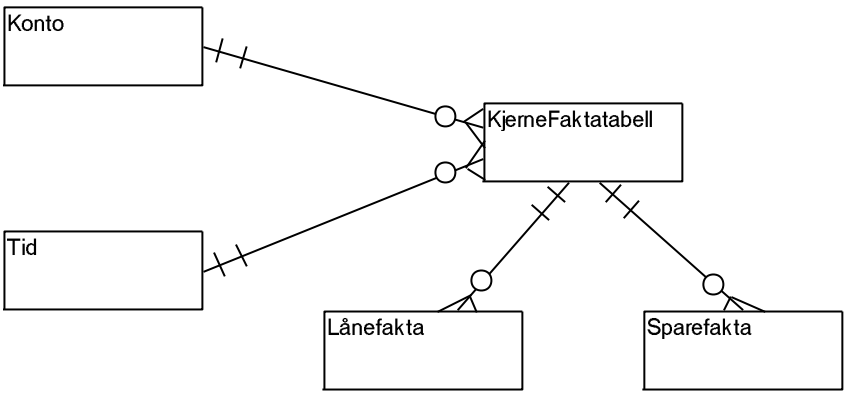
I modellen i figur 12.8 inneholder navigeringstabellen koden til den aktuelle avdelingen, her kalt *foreldreavdelingskode*, samt avdelingskoden til en vilkårlig avdeling (*underordnetavdelingskode*). Attributtet *antall nivåer* angir hvor mange nivåer det er fra foreldreavdeling til underordnet avdeling (for eksempel vil verdien 1 bety at den ene avdelingen er direkte underordnet den andre, mens verdien 0 betyr at vi har den avdelingen som er øverst i hierarkiet). Attributtet *lavestnivåflagg* er et boolsk attributt som er sant for en underordnetavdeling som er lavest i hierarkiet.

Denne strukturen gjør det mulig å bruke SQL-spørringer med GROUP BY på alle attributtene i Avdeling. På samme måte kan vi bygge et hierarki for produkt/deler.

11.3.2 Heterogene produkter

I det gjennomgående eksempelet vi har brukt, har vi en butikkjede med en enkelt type fakta, nemlig salg (som måles i antall og kroner). En del virksomheter har imidlertid så forskjelligartede produkter at det er vanskelig å bruke en enkelt faktatabell. Vi ser dette typisk innen bank og forsikring, som nå også i økende grad slås sammen. Banker har for eksempel både lånekontoer, brukskontoer, skattekontoer og sparekontoer. Mens forsikringsselskaper har forskjellige typer forsikring. For en bank vil det være problematisk å samle fakta om lånekontoer og brukskontoer i en faktatabell. Siden hver produkttype vil kreve egne, spesifikke attributter, vil faktatabellen inneholde en rekke nullverdier for hver post. Samtidig skal fakta knyttes opp mot den samme kundebasen.

Det er imidlertid heller ikke spesielt effektivt å ha en faktatabell for hver type produkt. Mange analyser vil trenge fakta som i så fall vil være spredt utover mange faktatabeller. Løsningen på dette problemet er å bruke en tabell med kjernefakta, og så knytte denne opp mot spesielle faktatabeller for hver produkttype. Resultatet kan være en modell som vist nedenfor.



Figur 11.9 Dimensjonsmodell med heterogene produkter

11.3.3 Kjeder og sirkler

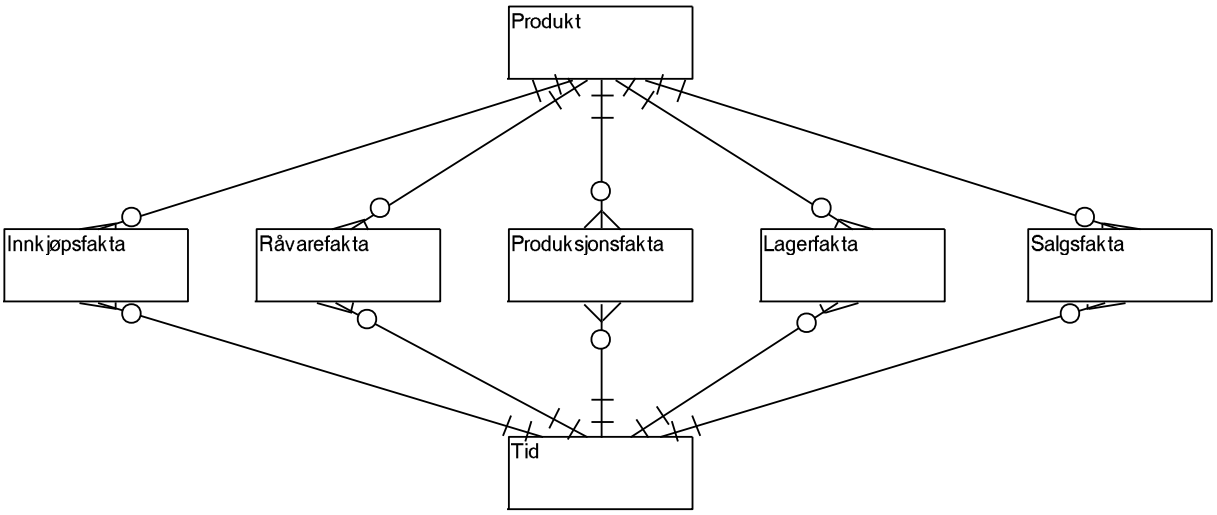
Innen mange virksomheter har man en kjede av aktiviteter som følger i en spesifikk rekkefølge. Vi kjenner slike som distribusjonskjeder og verdikjeder. Enten vi skal samle data internt (verdikjede) eller fra flere virksomheter (distribusjonskjede), er det logisk å bruke en faktatabell for hvert trinn i kjeden. Innen en produksjonsbedrift kan vi for eksempel opprette følgende faktatabeller:

- innkjøp
- lagerhold råvarer
- produksjon
- lagerhold ferdigvarer
- distribusjon
- salg

Data til disse faktatabellene kan komme fra flere forskjellige systemer, spesielt hvis man samler data fra hele distribusjonskjeden. En fordel med foretakssystemer er at virksomheten da vil kunne hente alle interne data fra samme database (samt at alle datadefinisjoner allerede er standardisert).

I en slik *kjede* med faktatabeller vil det være konforme dimensjoner som knytter dem sammen. I eksempelet ovenfor kan det for eksempel være produkt, Både innkjøp og lagerhold av råvarer vil da knyttes til det produktet de skal brukes til å lage. En slik dimensjonsmodell kan se ut som i figur 12.10.

I andre tilfelle kan man ikke sette opp klare kjeder som representerer rekkefølger av aktiviteter. For eksempel innen helsesektoren vil pasienter gå frem og tilbake mellom primærlege og sykehus, multiple diagnoser og behandlinger, med mulige overflytninger mellom avdelinger og til og med sykehus, kontroller etter hjemsendelse m.m. Forskjellige ledd vil være involvert for forskjellige pasienter, og rekkefølgen av dem vil også variere. I slike tilfelle sier vi at vi har en *sirkel* av faktatabeller.



Figur 11.10 En kjede av faktatabeller, her knyttet sammen av produktdimensjonen (og Tid)

Sirkler av faktatabeller er typisk i datavarehus for forskjellige typer virksomheter. Foruten helsevesenet kan vi nevne forsikring, apoteker og personalledelse.

11.3.4 Snapshots

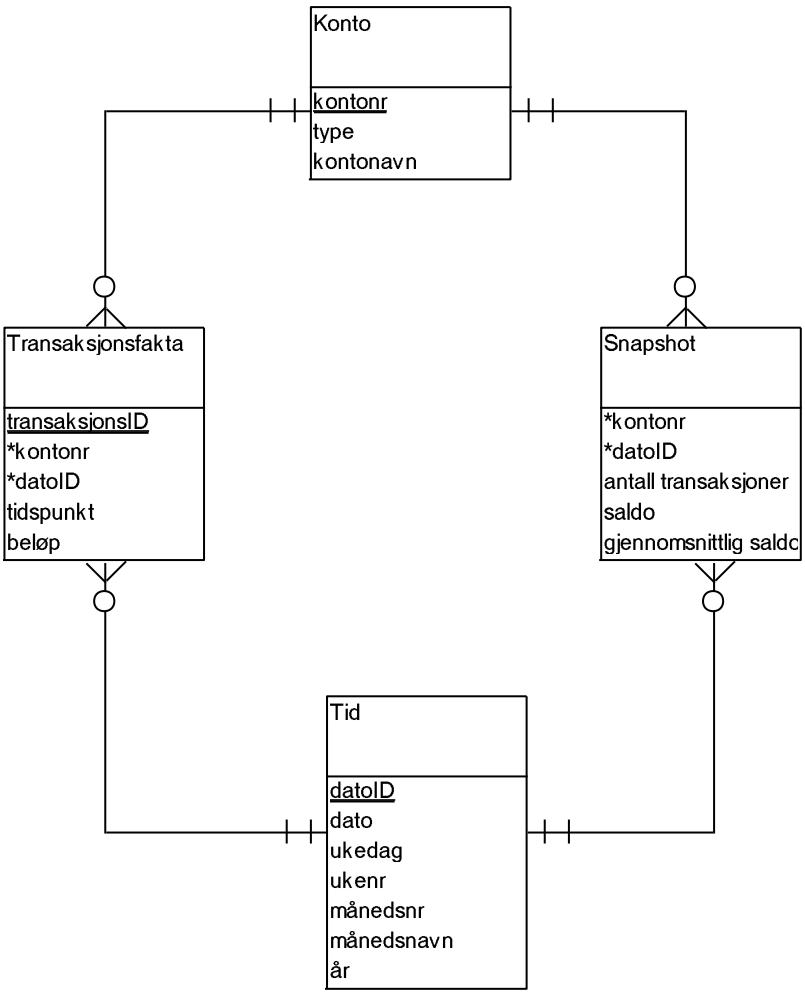
Den grunnleggende granulariteten for faktatabeller er transaksjonsdata. I tillegg til disse er det ofte at man har snapshottabeller i datavarehuset. Snapshottabeller trenger man når man i tillegg til transaksjonsdata trenger staturdata for bestemte tidspunkt. Et godt eksempel på dette er transaksjonsfakta for en bank. I denne faktatabellen vil hvert innskudd og hvert uttak være registrert som en post. For banken er det imidlertid også interessant å vite kontosaldo for eksempel for hver dag, uke eller måned. Dette kan selvsagt beregnes ved rapportering, men slike beregninger kan fort bli svært ressurskrevende. Det er derfor bedre å gjøre dem en gang for alle og legge data i en snapshottabell.

Figur 12.11 viser en tabell med transaksjonsfakta og en daglig snapshottabell. I tillegg til daglige snapshottabeller kan man ha egne ukentlige og månedlige tabeller. Det er også vanlig å bruke *rullerende* snapshot, der man har status for eksempel for hittil i måneden eller hittil i år.

> I mange datavarehusmodeller hører transaksjonsfakta og snapshotfakta sammen som pepper og salt. Transaksjonsfaktatabellene gir detaljert oversikt over virksomheten. Snapshottabellen gir status for et gitt tidspunkt.

11.3.5 Aggregerte tabeller

Aggregerte tabeller inneholder ferdig summerte data, typisk summeringer av transaksjonsfakta. Hensikten med aggregeringer er å spare tid og ressurser ved spørringer. Brukerne av et datavarehus etterspør oftest data som er summert på samme måte, og det er ofte de samme summeringene det blir spurt etter. Logikken bak et datavarehus er da at man foretar disse summeringene en gang for alle, og lagrer resultatet i en tabell. I datavarehussammenheng er det alltid et mål å effektivisere spørringer.



Figur 11.11 Transaksjonsfakta og snapshotfakta

Oppsummeringer kan skje på mange nivåer. I vårt gjennomgående eksempel med butikkjeden, vil typiske summeringer være salg per dag, salg av hver enkelt vare per dag og salg av hver enkelt vare i hver enkelt butikk per dag. Dessuten vil de samme summeringene per uke og måned kanskje være enda mer etterspurt.

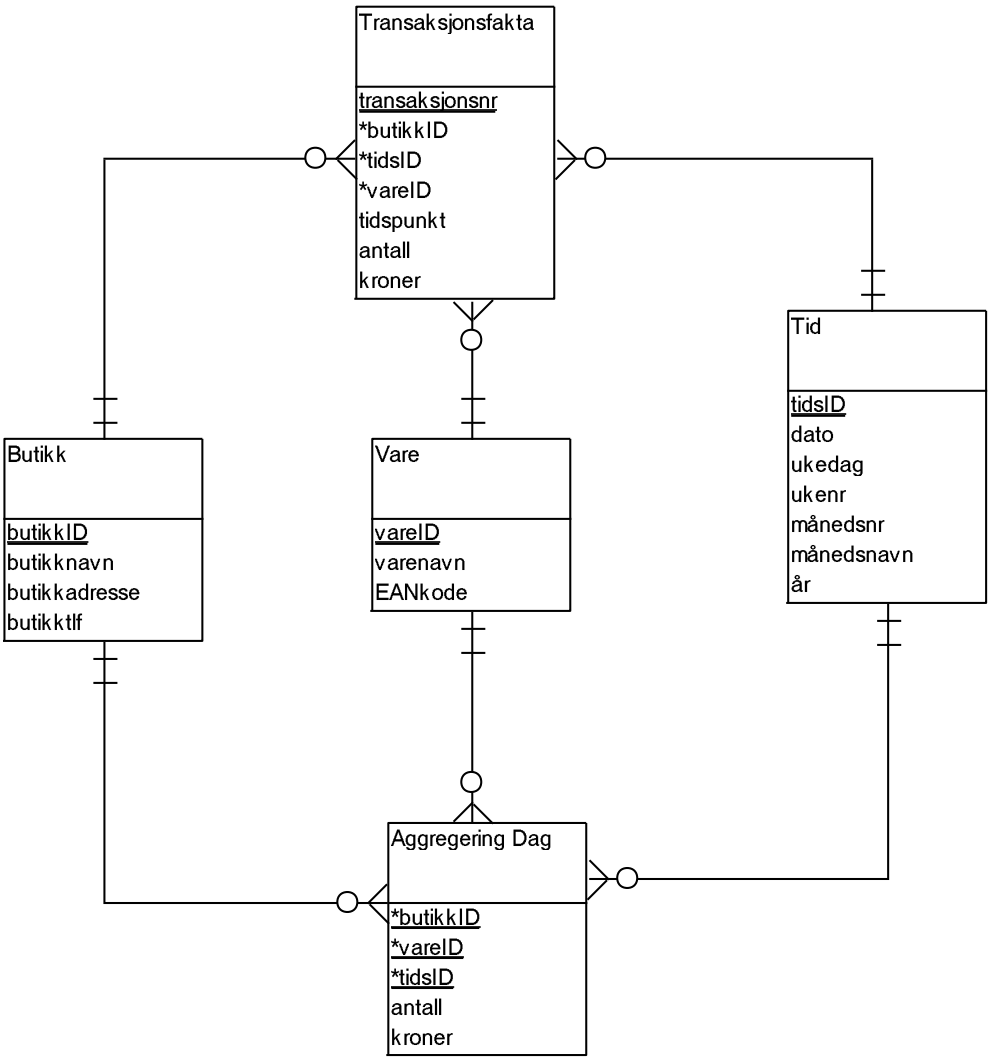
Til en tabell med transaksjonsfakta kan det dermed høre en rekke forskjellige aggregerte tabeller, med ferdig summerte data på forskjellige summeringsnivåer. I tillegg kommer en eller flere snapshottabeller. Figur 12.12 viser en samling aggregerte tabeller.

11.3.5 Dimensjoner med forskjellige roller i dimensjonsmodeller

I mange tilfelle kan vi opplede at vi har samme fremmednøkkel flere ganger i samme faktatabell, men i forskjellige roller. For et flyselskap opererer man for eksempel med ankomstflyplass og destinasjonsflyplass. I tillegg har man ofte transittflyplass. For en bestemt flight kan man dermed få flyplass tre ganger i tre forskjellige roller.

Problemet er at SQL ikke håndterer spørringer med tre forskjellige relasjoner mellom faktatabeller og flyplassdimensjonen. I så fall måtte fremmednøkkelen hatt samme verdi

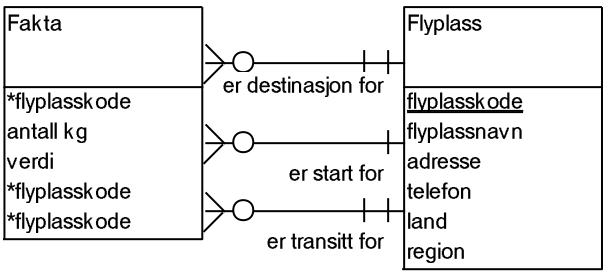
for alle tre utgavene i faktatabellen, og det skal den jo åpenbart ikke ha. Det er heller ikke noe særlig attraktivt alternativ å ha tre forskjellige flyplassdimensjoner, som alle inneholder de



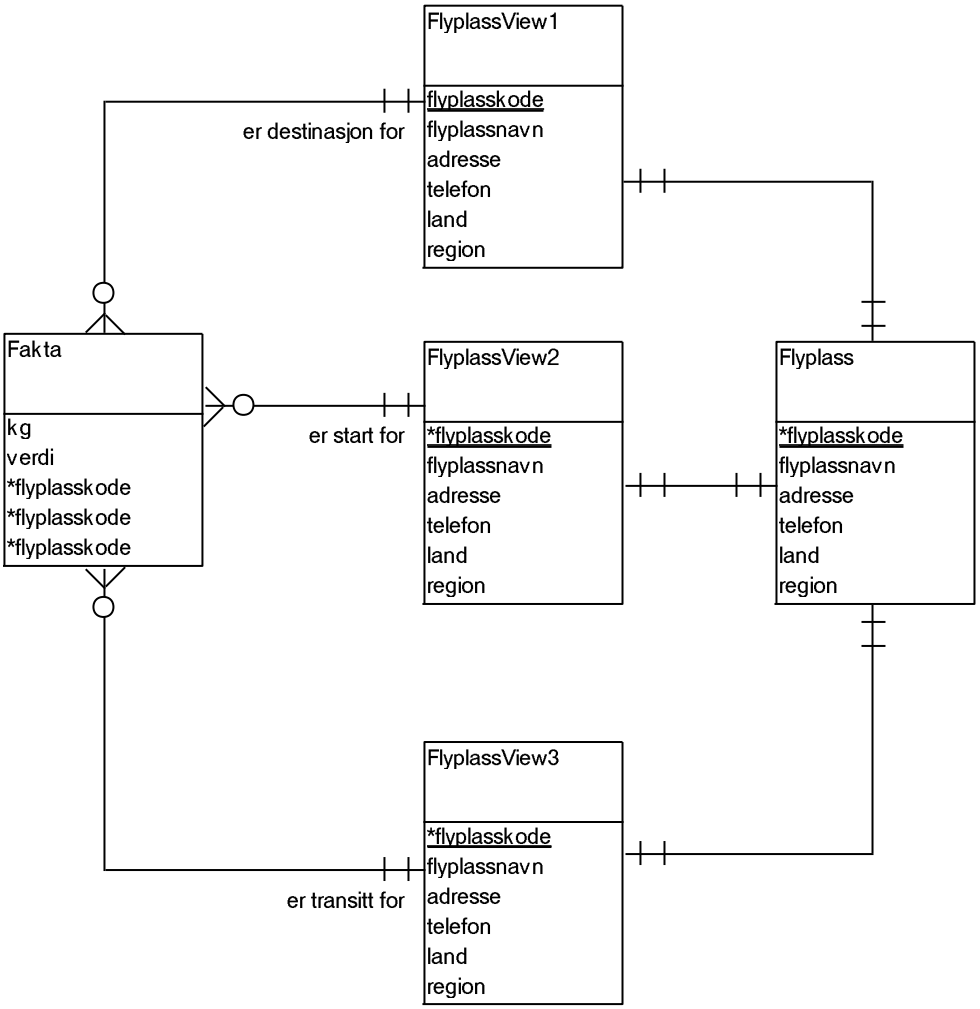
Figur 11.12 Transaksjonsfakta og én aggregert tabell

samme postene. Selv i datavarehussammenheng vil man helst unngå slike redundanssituasjoner.

Løsningen er å bruke views. For hver rolle lager man et view mot den underliggende dimensjonstabellen. Nå kan man uten problemer skrive SQL-spørringer som opererer med de tre rollene for flyplass.



Figur 11.13 Dimensjonsmodell med forskjellige roller for en dimensjon



Figur 11.14 Bruk av views i dimensjonsmodellen

Dato vil ofte inngå i flere roller i samme faktatabell.

Kapittel 12. Regulær oppdatering av datavarehuset

Oppdatering av datavarehuset vil typisk skje en gang i døgnet. Oppdateringen vil både gjelde faktatabeller og dimensjonsmodeller. For faktatabellen er det forholdsvis enkelt, der skal vi bare legge til de nye postene siden sist. For dimensjonsmodellene er det enkelt så lenge vi holder oss til type 1 oppdatering, men det blir straks litt mer komplisert hvis vi skal bruke type 2.

12.1 Nødvendige endringer i dimensjonstabellene

For å klargjøre dimensjonstabellene for type 2 oppdatering, må vi legge til to datofelt. Det ene skal vise lastedato, det andre "utgått dato". Feltet for utgått dato setter vi i utgangspunktet til 31. Desember 9999. Følgende skript sørger for dette:

Først legger vi til to datofelter i dimensjonstabellene:

```
USE coffeemerchant_dw;
ALTER TABLE dim_inventory
ADD COLUMN actual_date DATE;
ALTER TABLE dim_inventory
ADD COLUMN expiry_date DATE;

USE coffeemerchant_dw;
ALTER TABLE dim_employee
ADD COLUMN actual_date DATE;
ALTER TABLE dim_customers
ADD COLUMN expiry_date DATE;

USE coffeemerchant_dw;
ALTER TABLE dim_customers
ADD COLUMN actual_date DATE;
ALTER TABLE dim_customers
ADD COLUMN expiry_date DATE;
```

Deretter setter vi inn verdier:

```
USE coffeemerchant_dw;
UPDATE dim_customers
SET actual_date = CURRENT_DATE;
UPDATE dim_customers
SET expiry_date = '9999-01-31';
UPDATE dim_employee
SET actual_date = CURRENT_DATE;
UPDATE dim_employee
SET expiry_date = '9999-01-31';
UPDATE dim_inventory
SET actual_date = CURRENT_DATE;
UPDATE dim_inventory
SET expiry_date = '9999-01-31';
```

12.2 Oppdatering av faktatabellen

Før vi oppdaterer faktatabellen, må vi legge inn nye poster i kildesystemet. Vi nøyer oss med en ordre med et par ordrelinjer:

```
USE coffeemerchant;
INSERT INTO Orders
VALUES (214510, '2006-04-20', '2392-234', 34669,
4112);
```

```
USE coffeemerchant;
INSERT INTO OrderLines VALUES (214510, 5, 344, 9,
7, 0);
INSERT INTO OrderLines VALUES (214510, 6, 188, 15,
3.9, 0.15);
```

Vi kan nå velge flere metoder for å oppdatere faktatabellen, eller rettere stagingtabellen. Det enkleste er kanskje å ta ut ordrenummer som ikke allerede finnes:

```
USE coffeemerchant_dw;
TRUNCATE TABLE stage_order_fact;
INSERT INTO stage_order_fact
SELECT
  o.orderID,
  o.orderdate,
  o.consumerID,
  NULL,
  o.employeeID,
  ol.inventoryID,
  ol.quantity,
  ol.price,
  ol.discount
FROM coffeemerchant.orders o,coffeemerchant.orderlines ol
WHERE o.orderID = ol.orderID
AND o.OrderDate NOT IN (
  SELECT dato FROM fact_sales_dimensions);
```

Vi kan også velge ut på grunnlag av dato:

```
USE coffeemerchant_dw;

TRUNCATE TABLE stage_order_fact;
INSERT INTO stage_order_fact
SELECT
  o.orderID,
  o.orderdate,
  o.consumerID,
  NULL,
  o.employeeID,
  ol.inventoryID,
  ol.quantity,
  ol.price,
  ol.discount
FROM coffeemerchant.orders
o,coffeemerchant.orderlines ol
WHERE o.orderID = ol.orderID
AND o.OrderDate = CURRENT_DATE;
```

Hvis vi kjører skriptet etter midnatt (noe som er vanlig), skifter vi ut den siste linjen i spørringen med dette:

```
AND o.OrderDate = SUBDATE(CURRENT_DATE,1);
```

12.3 Oppdatering av dimensjonstabellene

Når vi skriver skriptene for oppdatering av dimensjonstabellene, må vi ta stilling til hvilken type oppdatering vi vil ha. Valg av type skjer for det enkelte felt i tabellene.

Først lager vi et script for å sjekke nye registreringer i kildesystemet. Her er brukt kundetabellen som eksempel:

```
USE coffeemerchant_dw;
INSERT INTO dim_customers
SELECT
  NULL,
  consumerID,
  firstname,
  lastname,
  street,
  zipcode,
  city,
  state,
  CURRENT_DATE,
  '9999-12-31'
FROM coffeemerchant.consumers c
WHERE c.consumerID NOT IN (
  SELECT customerID
  FROM dim_customers);
```

La oss si at vi ønsker type 1 oppdatering av kundetabellen. Vi lager da et script som sjekker om det er forandringer i kundedata. I kundetabellen er de aktuelle forandringene som kan skje knyttet til flytting, dvs ny adresse, postnummer, by og stat.

```
USE coffeemerchant_dw;
UPDATE dim_customers a,
       coffeemerchant.consumers b
SET a.street = b.street
WHERE a.customerID = b.consumerID
AND a.street <> b.street;
UPDATE dim_customers a,
       coffeemerchant.consumers b
SET a.city = b.city
WHERE a.customerID = b.consumerID
AND a.city <> b.city;
UPDATE dim_customers a,
       coffeemerchant.consumers b
SET a.zipcode = b.zipcode
WHERE a.customerID = b.consumerID
AND a.zipcode <> b.zipcode;
UPDATE dim_customers a,
       coffeemerchant.consumers b
SET a.state = b.state
WHERE a.customerID = b.consumerID
AND a.state <> b.state;
```

Tilsvarende script lager vi for de andre dimensjonene, dersom bare type 1 oppdatering skal gjøres.

Type 2 oppdatering gjør bruk av datofeltene. Feltet expiry_date settes da til datoen gammel post "går ut på dato". Deretter settes en ny post inn. Hvis vi igjen gjør bruk av tabellen kunde, vil vi sjekke på gatenavn og postnummer (ved flytting skulle dette holde). Scriptet vil nå se slik ut:

```
USE coffeemerchant_dw;
INSERT INTO dim_customers
SELECT
  NULL,
  consumerID,
  firstname,
  lastname,
  street,
  zipcode,
  city,
  state,
  '2006-04-20',
  '9999-12-31'
FROM coffeemerchant.consumers c
WHERE c.consumerID NOT IN (
  SELECT customerID
  FROM dim_customers);
```

```
INSERT INTO dim_customers
SELECT
  NULL,
  b.consumerID,
  b.firstname,
  b.lastname,
  b.street,
  b.zipcode,
  b.city,
  b.state,
  CURRENT_DATE,
  '9999-12-31'
FROM coffeemerchant.consumers b,dim_customers a
WHERE a.customerID = b.consumerID
AND a.street<>b.street
AND EXISTS (
  SELECT * FROM dim_customers x
  WHERE b.consumerID = x.customerID
  AND a.expiry_date = SUBDATE(CURRENT_DATE,1))
AND NOT EXISTS (
  SELECT *
  FROM dim_customers y
  WHERE b.consumerID = y.customerID
  AND y.expiry_date = '9999-12-31');
```

Kapittel 13 ETL og ETL-verktøy

Data i et datavarehus kommer fra et eller flere kildesystemer. Det første som skjer er derfor at data må hentes ut fra disse. Denne prosessen kalles dataekstraksjon, og er allerede gjennomgått. Deretter må dataene kontrolleres og summeres, og ofte også gjøres om. Til slutt skal de ferdig behandlede data lastes inn i datavarehuset og de tilhørende Data Marts. Hele prosessen kalles ofte ETL, for Ekstraksjon, Transformasjon og Lasting. I datavarehusets arkitektur snakker vi også ofte om Data Staging Area som den delen av systemet der transformasjon og lasting skjer.

Et Data Staging Area er i følge Ralph Kimball (Kimball 2013) *"A storage area and set of processes that can clean, transform, combine, deduplicate, household, archive, and prepare source data for use in the data warehouse*". DSA er altså både et lagringsområde og en samling prosesser.

Data Staging Area (DSA) er selve "arbeidsbenken" i datavarehuset (ibid.). Her hentes rådata inn, "renses", kombineres, transformeres, summeres, arkiveres og lastes inn i datavarehus og data marts. Et sentralt prinsipp for DSA er at det **ikke** skal være tilgjengelig for spørringer fra brukere. Spørringer skal alltid rettes mot presentasjonsserveren, dvs selve datavarehuset med tilhørende data marts.

Rådata kan lastes inn i en relasjonsdatabase før prosesseringen i DSA, noe vi skal gjøre med Pentaho Data Integration. Det er imidlertid ikke alltid dette er den beste løsningen. I situasjoner med svært store datamengder der det vil bli foretatt mange summeringer, vil det ofte være mer effektivt først å bruke flate filer. Dette fordi det er betydelig overhead i relasjonsdatabaser når man skal summere.

13.1 Oversikt over hva som skjer ved transformasjon

I følge Kimball vil DSA ofte være basert på normaliserte ER-modeller som gjenspeiler både kildesystemene og dimensjonsmodellene i datavarehuset. Dermed ligner hans definisjon av lagringsdelen av DSA mye på Inmons definisjon for detaljtabellene i datavarehuset, bortsett fra at Kimball åpner for bruk også av flate filer og proprietære lagringsformater. Inmon bruker ikke begrepet Data Staging Area.

Når data er hentet ut fra kildesystemene, gjennomgår de en rekke forskjellige typer transformasjoner for å bli forvandlet til data som kan presenteres for brukerne. Kimball gir en klassifikasjon av de forskjellige typene (Kimball 2013), gjengitt i tabell 13.1.

Type	Beskrivelse
Integrasjon	Lage surrogatnøkler, mappe nøkler fra et system til et annet.
Vedlikehold av dimensjoner med langsom forandring	Identifisere og håndtere forandrede verdier i dimensjonene. Dette er typisk en av funksjonene i verktøy for å bygge DSA. Det er egne prinsipper for håndtering av forandringer i dimensjonsverdier, noe som blir gjennomgått nærmere i et senere kapittel.

Kontroll med referanseintegritet	Referanseintegritet for innkommende data sjekkes før lasting inn i relasjonsdatabasen. Dette gir mulighet for korrigering før lasting. Uten slik korrigering vil ikke et RDBMS ta imot dataene.
Denormalisering og renormalisering	En typisk transformasjon er denormalisering av et hierarki at tabeller til en enkelt dimensjonstabell (jfr dimensjonen KUNDE i kapittel 16). I andre tilfelle kommer data inn som en denormalisert flat fil, og må renormaliseres (som med våre rådata)
Rensing	Data fra kildesystemene kan inneholde feil, som må oppdages og rettes.
Typekonvertering	Dette kan inkludere transformasjon fra for eksempel Number til Varchar, fra ett tegnsett (EBCDIC) til et annet (ASCII) eller mellom forskjellige datoformater.
Beregninger og avledninger	Dette er forskjellige omgjøringer som skal gjøre data mer presentable. Et eksempel er å gjøre om kundenavn fra bare store bokstaver til vanlig format med stor forbokstav
Aggregering	Summeringer av data på forskjellige nivåer
Kontroll av datainnhold	Dette er bruk av kontrollsummer, opptelling av radantall og andre kontroller for å kontrollere om innkomne data er rimelige.
Kontroll av prosesser	Opprettelse av logger for å kunne følge prosessene som skaper data i spesifikke rader i både dimensjonstabeller og faktatabeller
Verktøyspesifikke transformeringer	Mange av de verktøyene som brukes krever egne transformeringer
Nullverdier	Nullverdier er ikke uvanlige i kildesystemene, og de kan behandles forskjellig i forskjellige systemer. Et RDBMS bruker gjerne null, mens mange eldre systemer bruker defaultverdier som 99999999
Avslutninger av prosesser	Alle prosesser i DSA kan ha behov for å kalle eksterne prosesser både før og etter kjøring. Et eksempel på dette er eksterne sorteringsprogrammer som kjøres for summering

Tabell 13.1 Oversikt over transformasjoner i DSA

13.2 Jobbkontroll i DSA

Siden det i Data Staging Area er en rekke prosesser som skal kjøres, er det viktig å kunne kontrollere dem. Typisk er at prosesser må kjøres i en bestemt rekkefølge. For eksempel vil rensing av data og kontroll med nullverdier måtte foretas først, sortering foretas før summering og oppdatering av dimensjonstabeller foretas før innhenting av nye faktaposter.

Typiske tjenester i jobbkontrollen er oppsummert i tabell 13.2 (Kimball 2013).

Tjeneste	Beskrivelse
Jobbdefinisjon	Trinnene i en jobb må kunne defineres på en eller annen måte. Disse definisjonene beskriver arbeidsflyten i datavarehuset. Mange verktøy for å lage datavarehus har egne funksjoner for å definere slik flyt.
Jobbtimeplan	Det må være mulig å sette tidspunkter de forskjellige jobbene skal starte.
Overvåkning	Systemet må ha muligheter for å overvåke hvilke jobber som er kjørt, kjører og skal kjøres.
Logging	Alle DSA-prosessene logges til tekstfiler slik at man i ettertid kan følge dem i detalj.
Unntakshåndtering	Det må defineres prosesser for å håndtere avvikende data. Siden selve prosessene i DSA skjer automatisk, er en vanlig måte å skrive avvikende data til en egen tabell der de kan kontrolleres manuelt i ettertid.
Feilhåndtering	Her dreier det seg ikke om feil i data, men tekniske feil som kan oppstå under kjøring. Håndtering av dette omfatter recovery, stopp og restart.
Varsling	Systemet må automatisk varsle den ansvarlige dersom noe går galt, for eksempel ved å sende en e-mail.

Tabell 13.2 Jobber i DSA

13.3 Bruk av ETL-verktøy

Med ETL-verktøy kan ETL-prosessen utføres i opp til tre trinn (Anda 2002):

1. Ekstrahering fra datakilde, enten flate filer eller relasjonsdatabase.
2. Nedlasting av data til stagingområde
3. ”Vasking” og transformering til datavarehus

Vi kan bruke 1, 2 eller alle tre trinnene. Gjør vi prosessen som ett trinn, lastes data direkte fra kilde til datavarehustabeller. Med alle tre trinn gjør vi bruk av egne stagingtabeller som mellomstadium før vaskede og transformerte data lastes i datavarehuset.

Det sentrale begrepet i ETL er mapping. En mapping beskriver en serie operasjoner som trekker data ut fra kildene, transformerer dem og laster den inn i måltabellene (Oracle 2001). Det å definere mappinger inkluderer dermed svært mye av det som regnes som innholdet i Data Staging Area.

Alle mappinger er basert på mappingoperatorer, som er logiske representasjoner av objekter i metadatabasen. En mappingoperator består ar radmengder (row sets), som kan være null eller flere rader med strukturerte data.

Det finnes mange leverandører av ETL-verktøy. Prinsippet for alle disse er at utviklerne arbeider med et grafisk brukergrensesnitt for å definere ETL-prosessene. Verktøyet genererer så koden for prosessene. Et eksempel på hvordan dette kan se ut er vist nedenfor, der vi ser hvordan mapping mellom kildetabell og datavarehustabell defineres i Pentaho Data Integration.

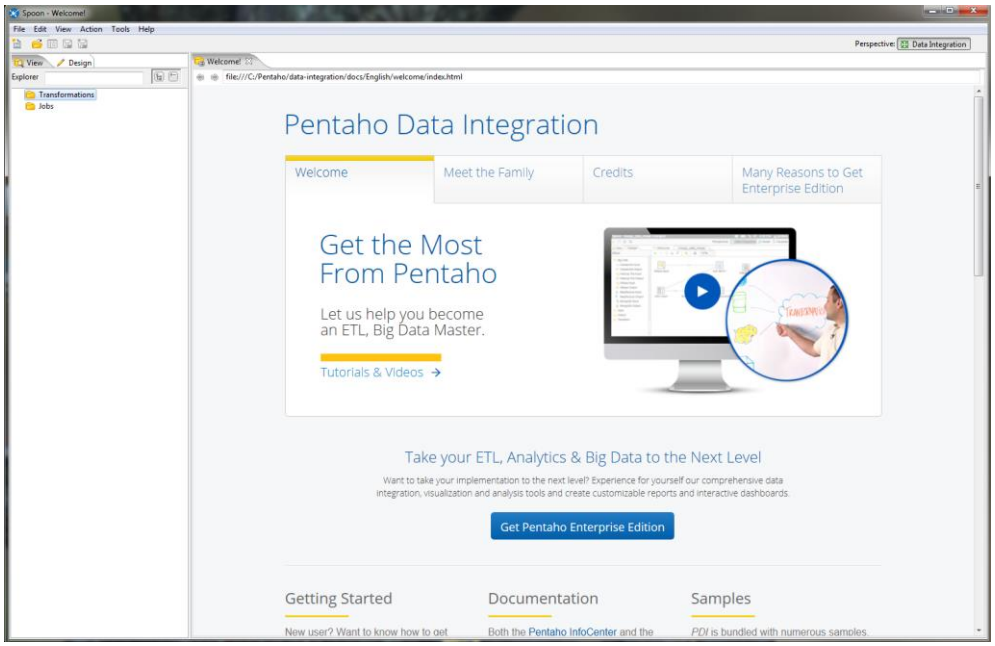
13.4 Pentaho Data Integration

I 2006 kjøpte Pentaho Kettle-prosjektet, som hadde utviklet et åpen kildekode ETL-verktøy. Kettle ble så integrert i Pentahos familie av BI-verktøy, etter hvert under det nye navnet Pentaho Data Integration. Dette består av tre deler: Spoon, som er selve modelleringsverktøyet, Pan og Kitchen. De to sistnevnte kjører de transformasjonene og jobbene vi definerer i Spoon. Det er en Java-applikasjon, som kommer som en zip-fil. Filen pakkes ut i en mappe (på min maskin har jeg laget en mappe Pentaho). Programmet startes med bat-filen Spoon.bat, som nå skal ligge i mappen Pentaho\data-integration.

Pentaho ble i 2015 kjøpt opp av Hitachi Data Systems, og er nå en del av firmaet Hitachi Vantara.

13.4.1 Komme i gang med Data Integration

PDI lastes ned som en zip-fil. Denne pakkes opp direkte under C:\. I mappen finnes en bat-fil som heter spoon.bat, og som starter PDI. Når vi starter Data Integration, får vi først innloggingsbildet:



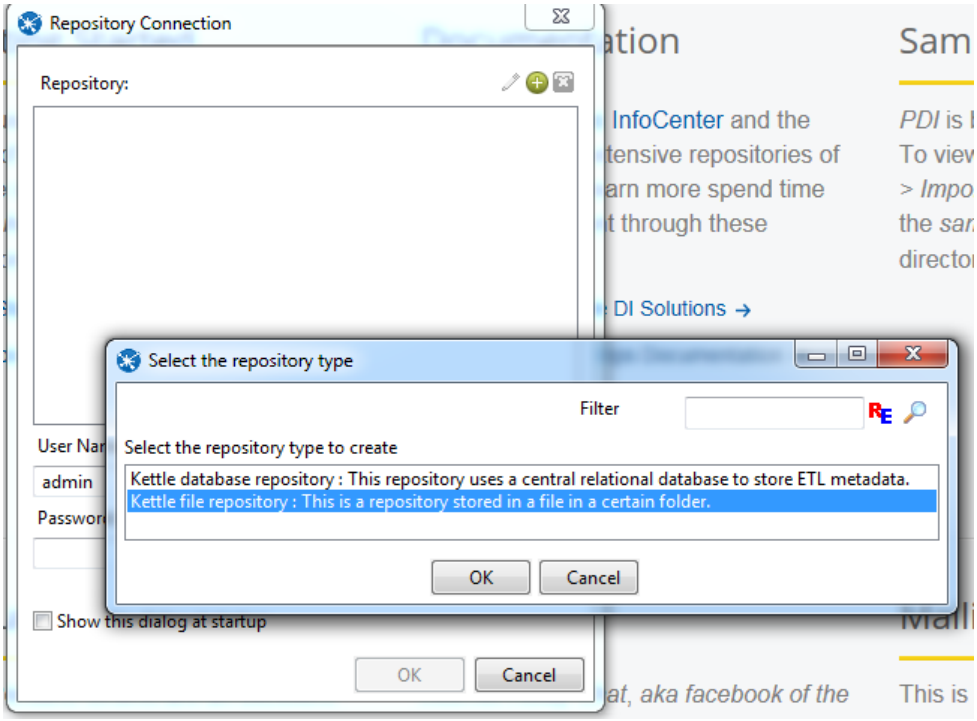
Figur 13.1 Pentaho Data Integration

Under **Getting Started** ligger en kortfattet (40 sider) innføring i bruken av Pentaho Data Integration (Get started with DI). Under Documentation ligger lenker til detaljert informasjon.

I DI kan vi arbeide med to forskjellige typer modeller: transformasjoner og jobber. En transformasjon er det samme som en mapping med tilhørende endringer av data. Det er her vi spesifiserer hva som skal gjøres med data. Den ferdige modellen vil bestå av en rekke forskjellige transformasjoner. Disse består av steg som skal utføres i en bestemt rekkefølge.

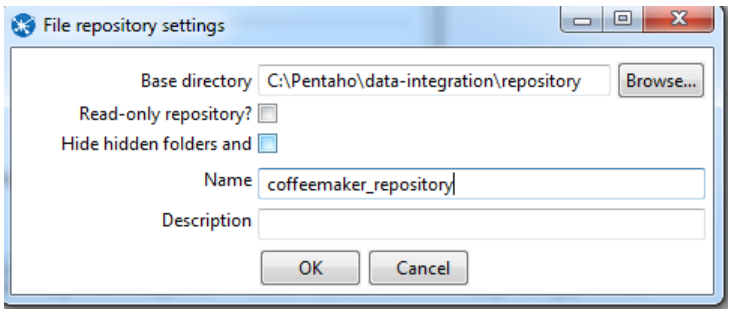
En jobb er en definisjon av hva hvilke transformasjoner som skal utføres, og i hvilken rekkefølge..

Før vi gjør noe annet, må vi lage et repository. Dette er en database der definisjonene vi gjør, blir lagret. Vi kan velge om vi vil lage et skjema i MySQL, eller om vi skal la DI bruke flate filer. Jeg har her valgt det siste:



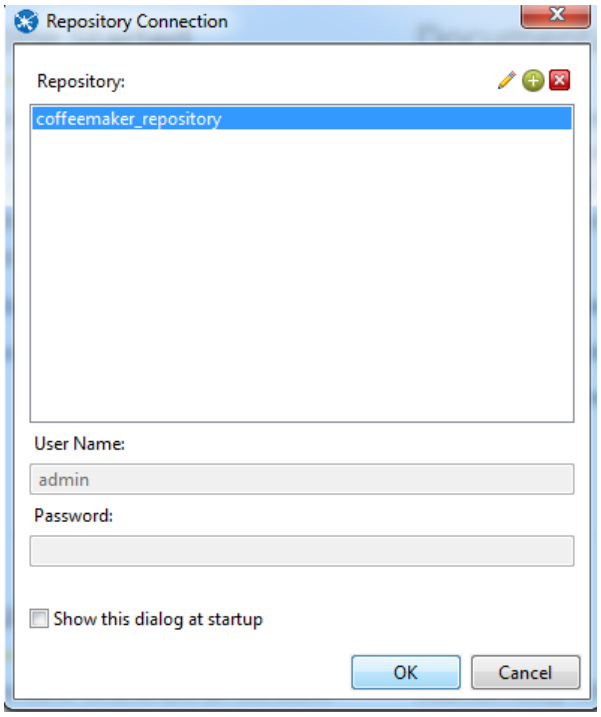
Figur 13.2 Angi plassering av Repository

Vi kan lage en mappe kalt repository under Pentaho\data-integration. Vi angir nå denne som sted for repository. Videre må vi gi det et navn, og en beskrivelse (i skjermbildet nedenfor er ikke description fylt ut, men det må gjøres).:



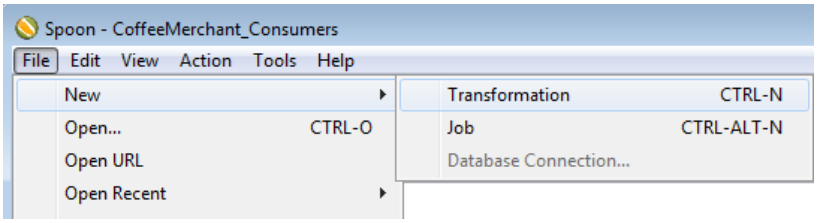
Figur 13.3 Angi egenskaper for Repository

Når vi nå klikker på OK, får vi opp en oversikt over våre repositoryer:



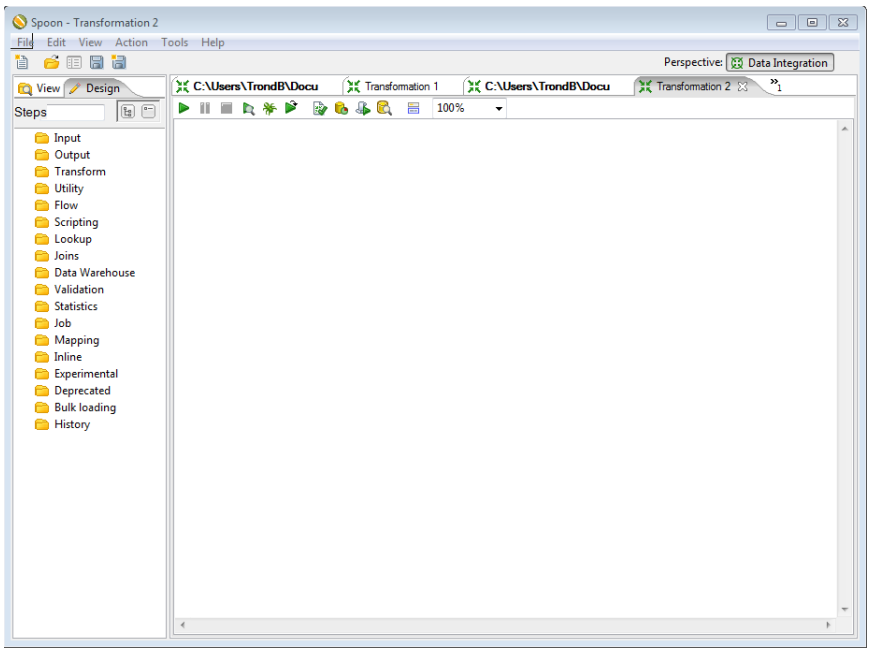
Figur 13.4 Oversikt over Repositoryer

Vi skal nå begynne på å lage en transformasjon. Vi velger File | New og deretter Transformation:



Figur 13.5 Opprette ny transformasjon

Dette bringer oss til designvinduet for transformasjoner, som ser slik ut:

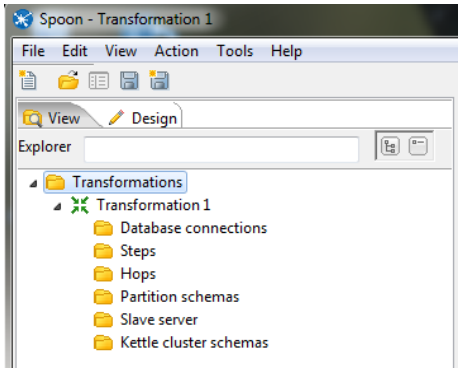


Figur 13.6 Designvinduet for transformasjoner

I vinduet til venstre finner vi mapper med verktøy for de forskjellige trinnene i en transformasjon. Vi begynner først med Input. Vi skal først gjøre det enkelt ved å lese tabeller i kildesystemet CoffeeMerchant og og bruke disse dataene til å lage dimensjonstabeller i datavarehuset. Til dette formålet må vi først lage en database kalt CoffeeMerchant_dw (eller noe annet). Transformasjonen fra f. eks. kundetabellen i kildesystemet til kundedimensjonen i datavarehuset består nå av følgende trinn:

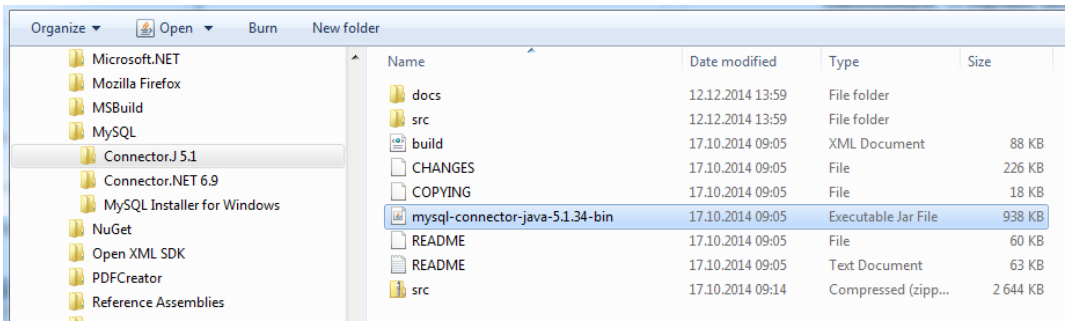
- Hent ut data fra kundetabellen i kildesystemet (Ekstraksjon)
- Legg til en surrogatnøkkel (Transformasjon)
- Legg inn i kundedimensjonen i datavarehuset (Lasting)

Det første vi må gjøre er imidlertid å lage databaseforbindelser. Vi trenger én til selve datavarehuset og en for hver relasjonsdatabase som skal være input. Vi definerer slike forbindelser ved å åpne fanen View øverst til venstre:



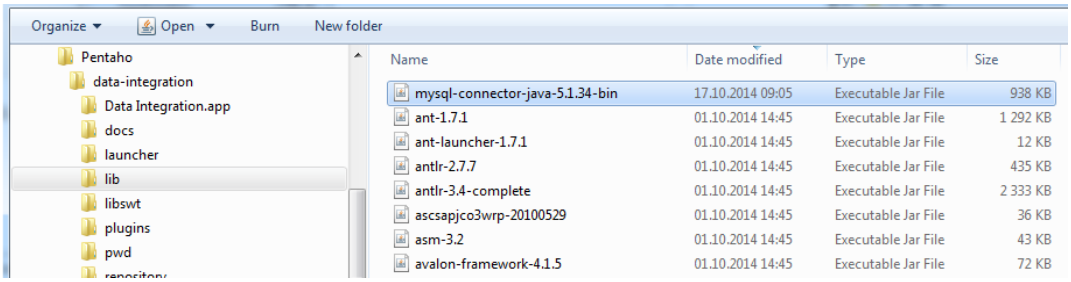
Figur 13.7 Utforskervinduet for transformasjoner

Men før vi kommer så langt, må vi kopiere JDBC for MySQL inn i mappen lib under Pentaho/data-integration. Har vi installert MySQL Connector/J, ligger den i en egen mappe under mappen MySQL i Programfiler (x86):



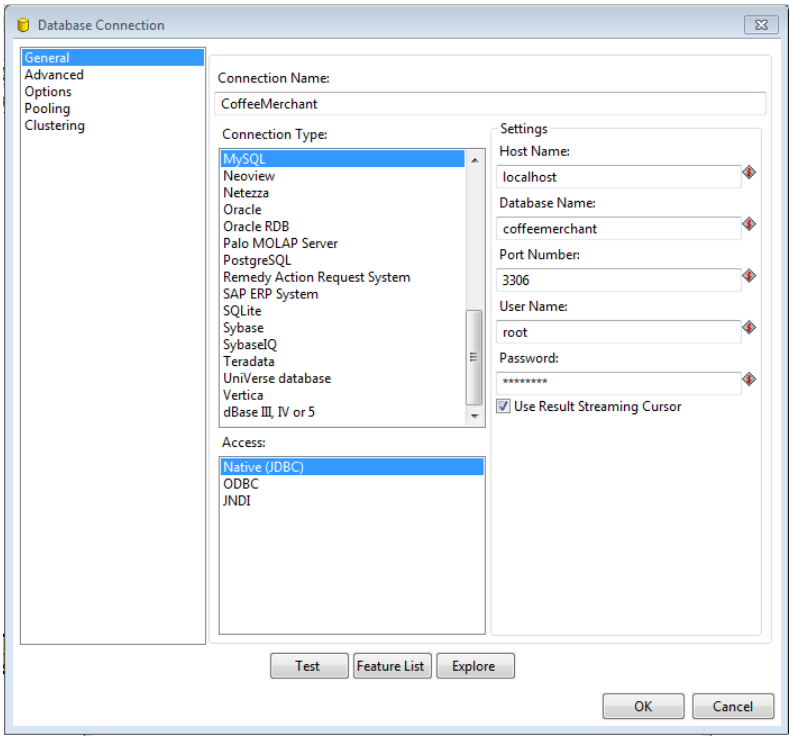
Figur 13.8 Standard plassering av MySql JDBC-driver

Vi kopierer denne til Pentaho/data-integration/lib:



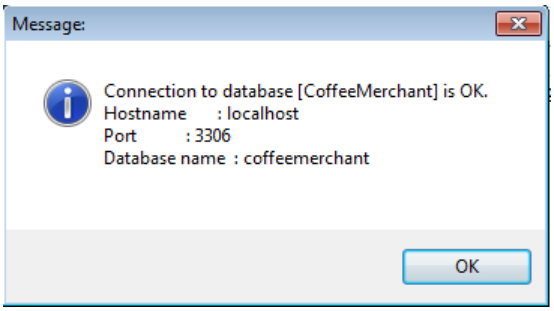
Figur 13.9 Plassering av driveren i Pentaho-mappene

Nå kan vi lage en ny forbindelse, og vi får da opp dette bildet:



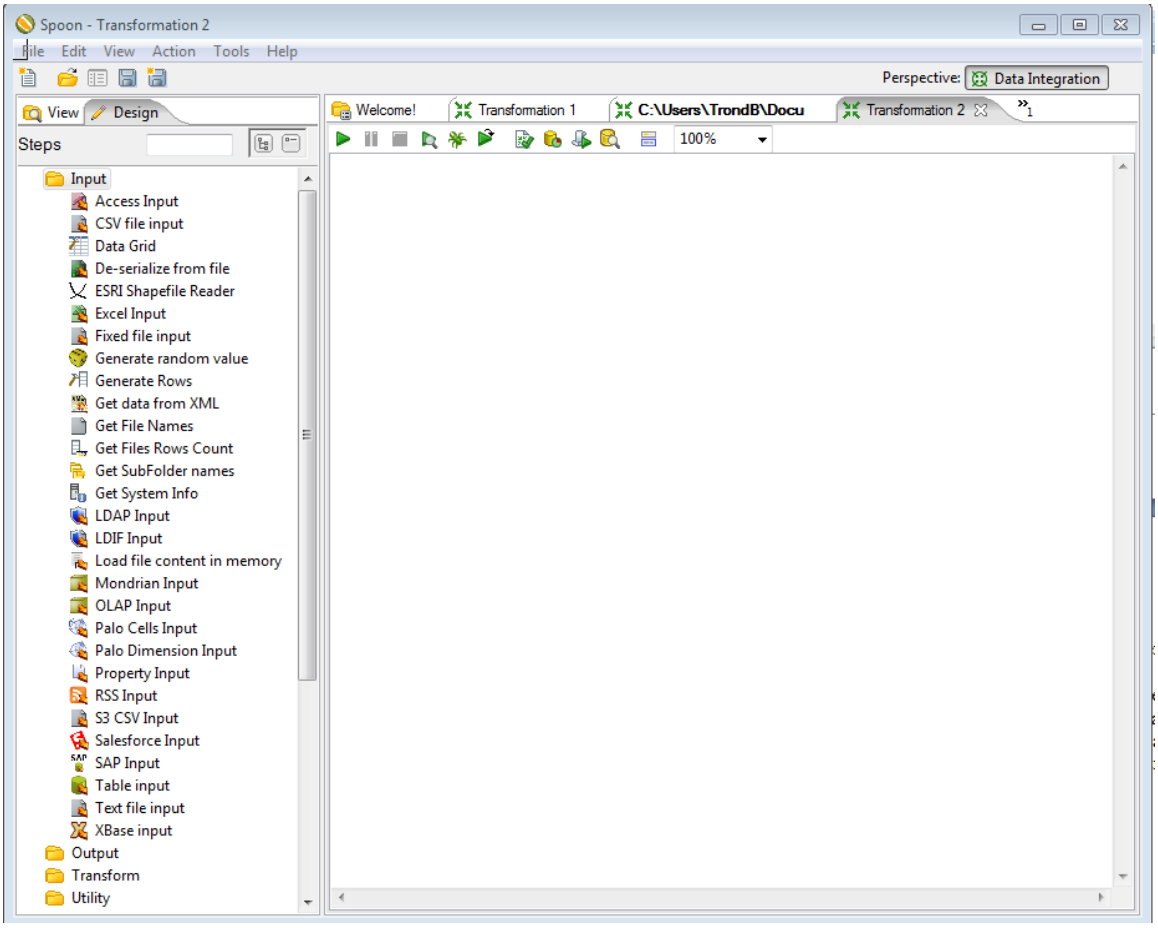
Figur 13.10 Databasevinduet

Her er det valgt en MySQL-database der coffeemerchant-databasen er installert. Vi bruker JDBC som mellomvare. Det kan være greit å teste forbindelsen (knapp nederst til venstre). Er alt i orden, får vi denne meldingen:



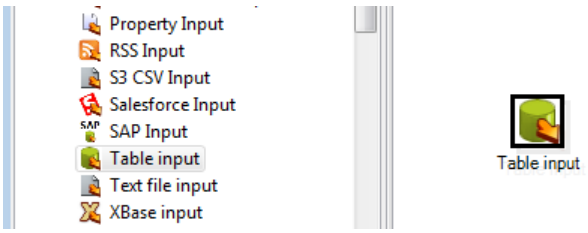
Figur 13.11 Vellykket test av databaseforbindelse

For å modellere en ETL-prosess, starter vi med å åpne mappen Input:



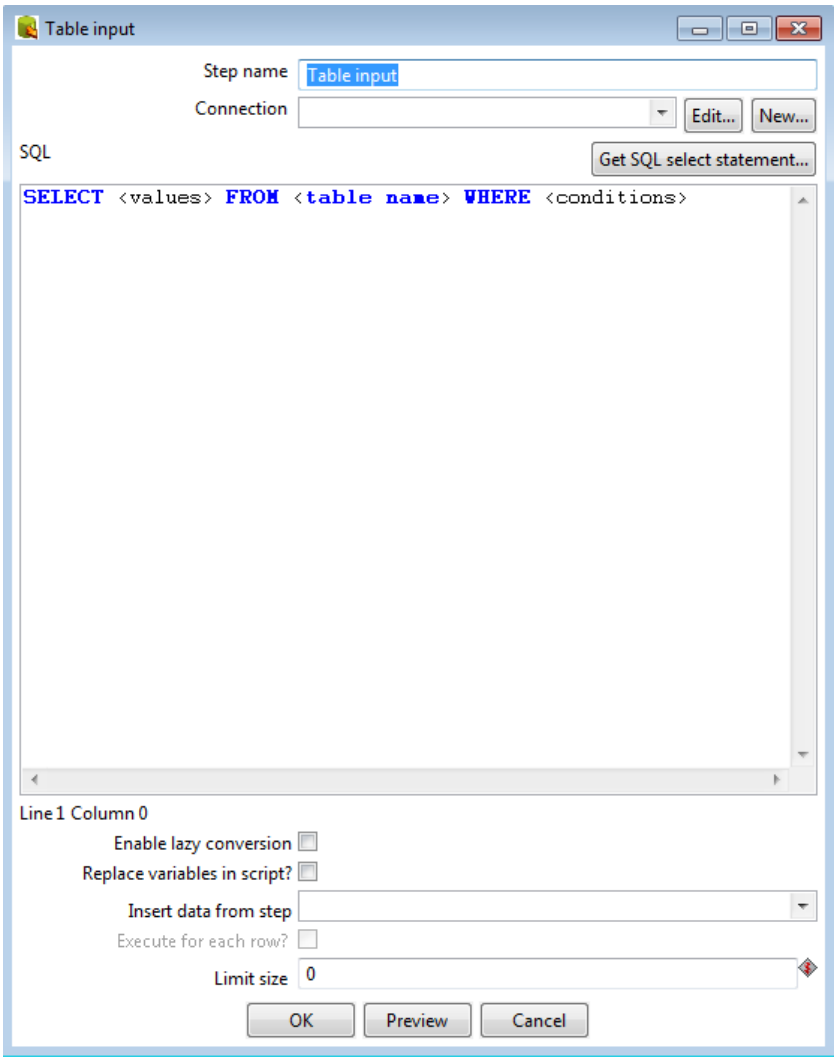
Figur 13.12 Innholdet i Input-mappen

Av verktøyene her skal vi starte med **Table input** (tredje valg nedenfra). Vi klikker på dette valget og ”dra” det deretter ut i arbeidsflaten:



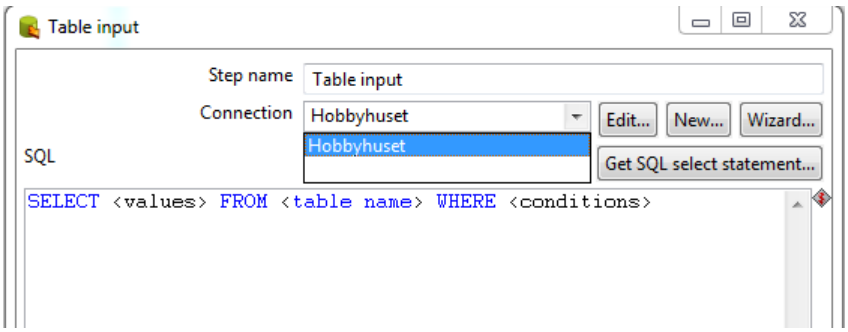
Figur 13.13 Tabellinput er valgt

Neste steg er å definere hva vi skal ha. Vi høyreklikker på ikonet for Table input, og velger **Edit step**. Vi får da frem et vindu der vi definerer både databaseforbindelsen og SQL-spørringen som skal til for å ekstrahere data:



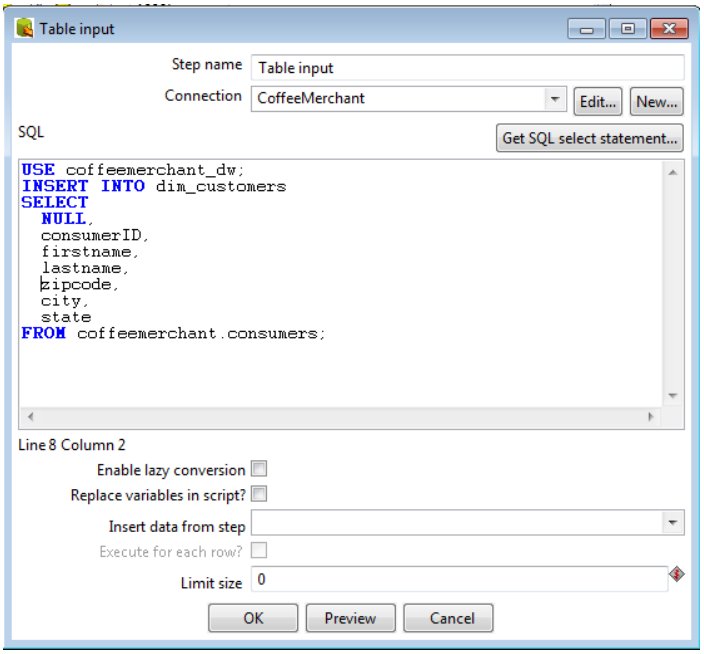
Figur 13.14 SQL-vindu for tabellinput

Vi må først velge databaseforbindelsen. Det gjør vi ved å klikke på Connection og velge den databaseforbindelsen vi skal bruke:



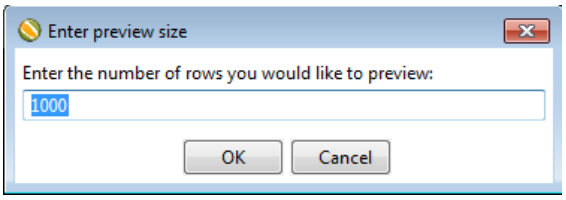
Figur 13.15 Valg av databaseforbindelse

Når alt er i orden, kan vi definere SQL-spørringen for å hente ut de data vi ønsker. La oss si at vi skal ha feltene kundeID, fornavn, etternavn, postnummer, by og stat. Dersom man er stø i SQL, kan man skrive en SELECT-setning selv:



Figur 13.16 Mulighet for å skrive egne SQL-utsagn

Trykker vi Preview får vi se om utvalget virker etter hensikten. Siden det kan være mange tupler i en tabell, spør Spoon først etter hvor mange den skal vise. Standard er 1000:



Figur 13.17 Mulig å velge antall rader som skal vises

Slik ser resultatet av spørringen ut:

customer_sk	customerID	FirstName	LastName	Zipcode	City	State
1	30121	F. Stanley	Best	72202	Little Rock	AR
2	30125	Duane A.	Maul	78155	Seguin	TX
3	30129	Alan J.	Rigas	07114	Newark	NJ
4	30132	T. Peter	Murray	48867	Owosso	MI
5	30136	Carl E.	Shelton	73102	Oklahoma City	OK
6	30139	David	Golkin	85012	Phoenix	AZ
7	30142	Jack R. Jr.	Kostantaras	53172	South Milwaukee	WI
8	30144	Terrence	Gerson	20005	Washington	DC
9	30147	Patrick J.	Townes	55439	Edina	MN
10	30148	Frank A.	Montrone	94304	Palo Alto	CA
11	30153	Robert	Choate	11788	Hauppauge	NY
12	30155	Dennis P.	Crist	53403	Racine	WI
13	30158	Richard E.	Huff	01581	Westborough	MA
14	30163	John M.	Hart	44118	Cleveland Heights	OH
15	30164	Alex W.	Hill	19464	Pottstown	PA
16	30168	John D.	McMeel	68124	Omaha	NE
17	30170	Lynn H.	Crosley	80222	Denver	CO
18	30174	L. H.	Harber	37214	Nashville	TN
19	30177	Roger A.	Bauer	28053	Gastonia	NC
20	30183	James D.	Garcia	24017	North West Roanoke	VA
21	30186	Robert R.	Graf	95066	Scotts Valley	CA
22	30190	Clifford M.	Tobin	2915	East Providence	RI

Figur 13.18 Resultatet vises

Transformasjonen består i dette tilfellet av å legge til en surrogatnøkkel. Det lar vi MySQL ta seg av, ved at vi har definert et felt customer_sk som er en integer med AUTOINCREMENT:

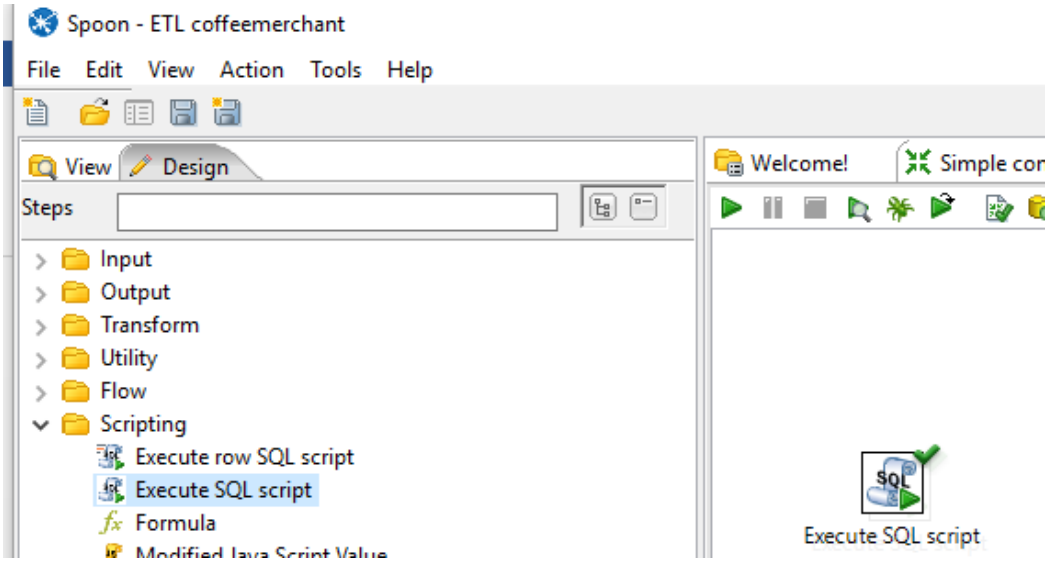
```
CREATE TABLE dim_customers (
    customer_sk INT NOT NULL AUTO_INCREMENT PRIMARY KEY,
    customerID  INT,
    FirstName   VARCHAR(30) NOT NULL,
    LastName    VARCHAR(30) NOT NULL,
    Zipcode     VARCHAR(5),
    City        VARCHAR(50),
    State       VARCHAR(2))
FROM coffeemerchant.consumers;
```

Vi skal nå se på forskjellige måter vi kan lage dimensjonstabellene på.

Vi skal først lage en transformasjon der vi bruker de scriptene vi allerede har. Dermed ser vi hvordan flyten i en transformasjon blir seende ut.

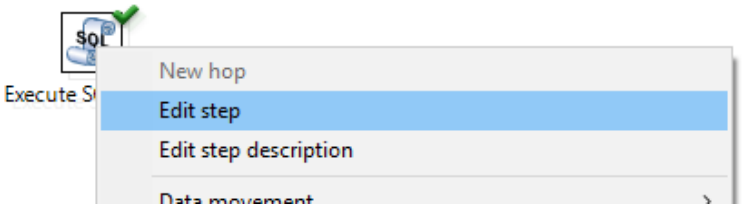
13.1.2 Opprette dimensjoner

Her skal vi lage SQL script for å opprette dimensjonstabellene. Vi velger *Execute SQL script* fra mappen *Scripting* ved å dra valget ut på arbeidsflaten:



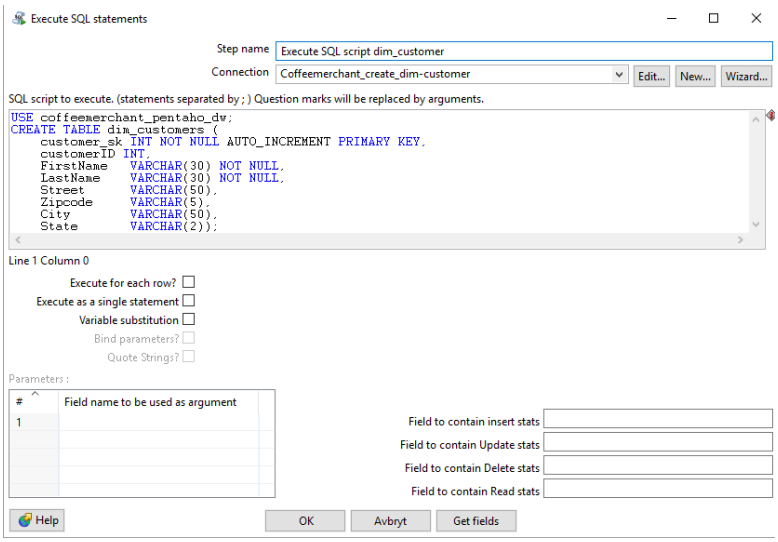
Figur 13.19 Opprette script i Spoon

Deretter kopierer vi inn et script, for eksempel for å opprette en kundedimensjon. Merk at jeg har opprettet et nytt skjema kalt **coffeemerchant_pentaho_dw** for dette datavarehuset. Vi høyreklikker på ikonet og velger Edit step:



Figur 13.20 Klar til skriving

Så limer vi inn scriptet:



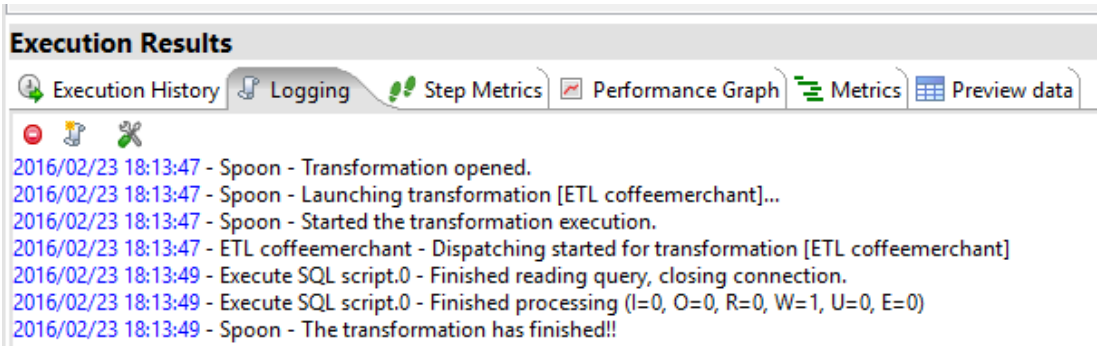
Figur 13.21 Scriptet er ferdig

Her har jeg også gitt steget et eget navn i stedet for det generiske Execute SQL script.

Vi kjører scriptet ved å klikke på den grønne pilen til venstre i menylinjen:

Figur 13.22 Klar til kjøring

I resultatvinduet får vi nå melding om hvordan det gikk:

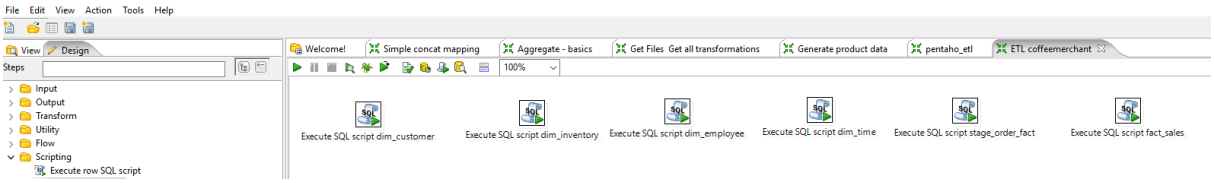


Figur 13.23 Scriptet er ferdig kjørt

Sjekker vi nå i MySql vil vi se at tabellen er opprettet.

Vi fortsetter nå med de øvrige dimensjonene.

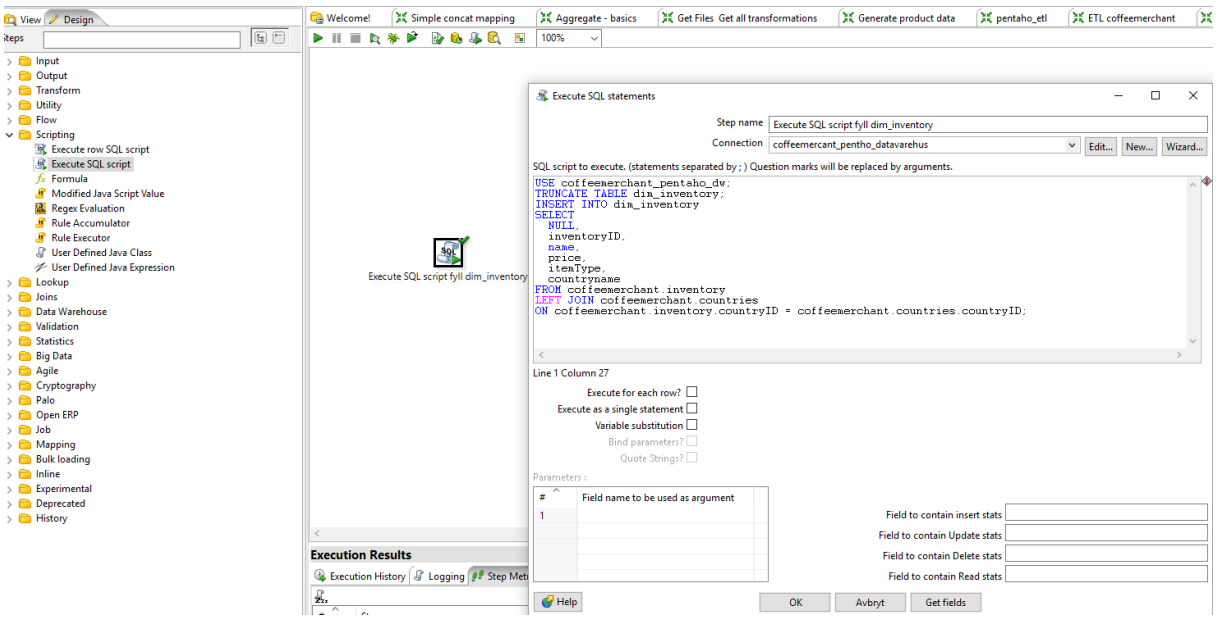
Når alle create-stegene er lagt inn, kjører vi dem (vi kan kjøre dem hver for seg etter hvert som vi lager dem, men da får vi feilmelding på de tabellene som allerede er laget):



Figur 13.24 Alle scriptene ligger klare i Spoon

13.1.3 Fylle dimensjonene

Vi kan først fylle dimensjonene ved å utføre de samme scriptene som vi tidligere har gjort med ren SQL. Her er vist hvordan vi kan definere en SQL script utførelse:



Figur 13.25 Script for å fylle tabell

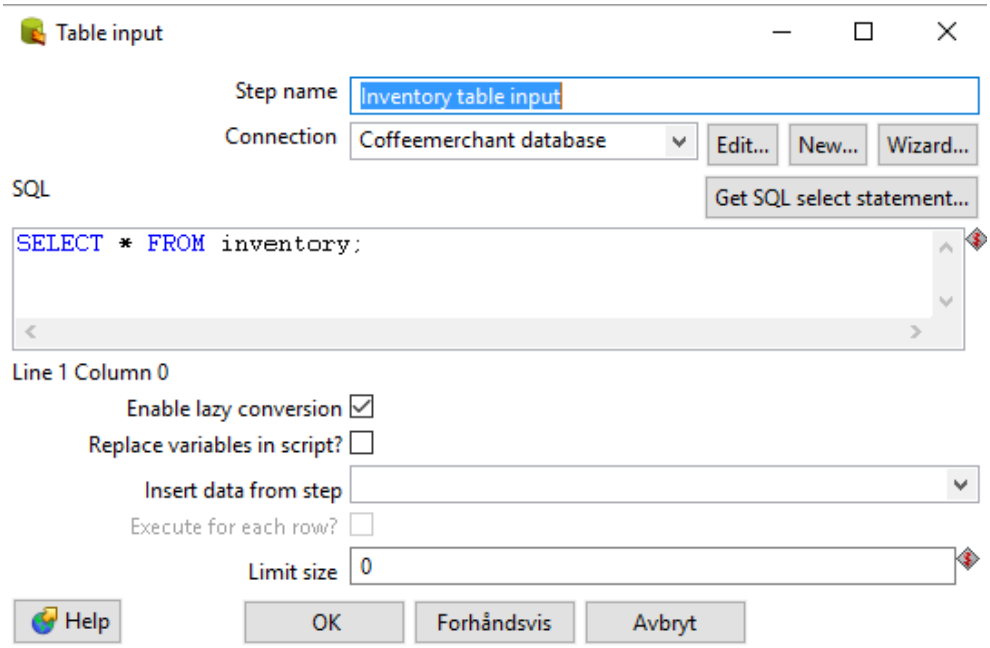
Resultatet, når vi kjører dette trinnet, er at vi fyller dimensjonstabellen med data:

Result Grid | Filter Rows: | Edit: | Export/Import: | Wrap Ce

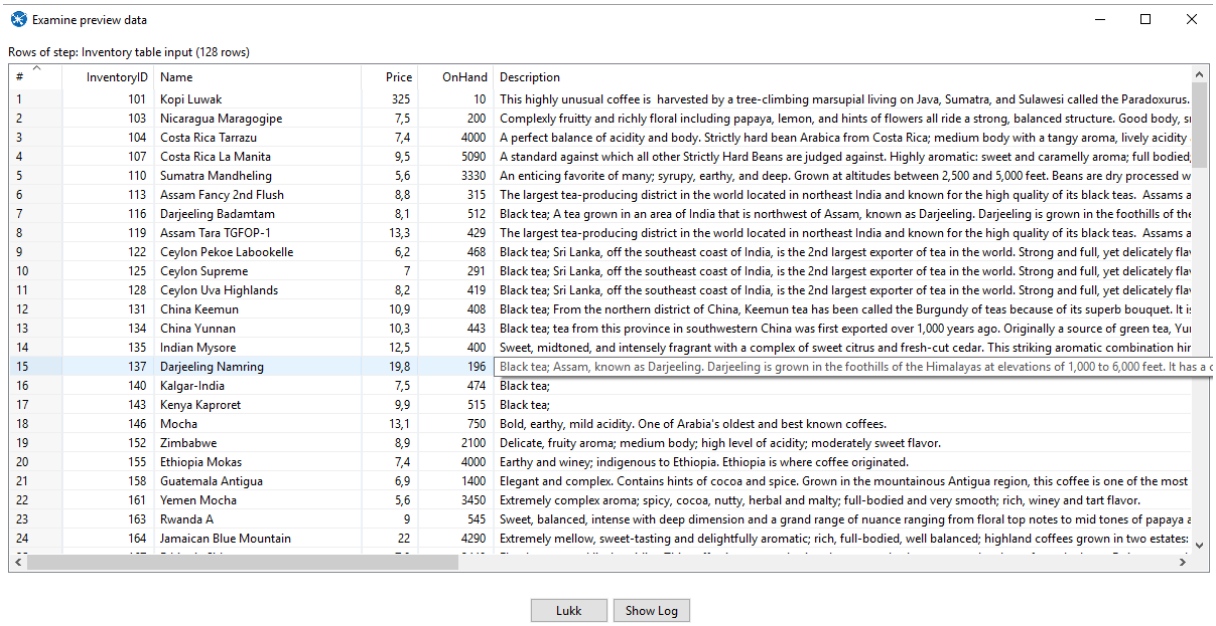
inventory_sk	InventoryID	Name	Price	itemType	countryname
1	101	Kopi Luwak	325.00	C	Indonesia, Republic of
2	103	Nicaragua Maragogipe	7.50	C	Nicaragua
3	104	Costa Rica Tarrazu	7.40	C	Costa Rica
4	107	Costa Rica La Manita	9.50	C	Costa Rica
5	110	Sumatra Mandheling	5.60	C	Indonesia, Republic of
6	113	Assam Fancy 2nd Flush	8.80	T	China
7	116	Darjeeling Badamtam	8.10	T	India
8	119	Assam Tara TGFOP-1	13.30	T	China
9	122	Ceylon Pekoe Labookelle	6.20	T	Sri Lanka
10	125	Ceylon Supreme	7.00	T	Sri Lanka
11	128	Ceylon Uva Highlands	8.20	T	Sri Lanka
12	131	China Keemun	10.90	T	China
13	134	China Yunnan	10.30	T	China
14	135	Indian Mysore	12.50	C	India

Figur 13.26 Ferdig fylt tabell

Vi skal nå lage transformasjoner for å fylle dimensjonstabellene. Vi går da til mappen Input og velger Input Table: Når vi har plassert ikonet på arbeidsflaten, høyreklikker vi, velger Edit og lager en select * -setning:

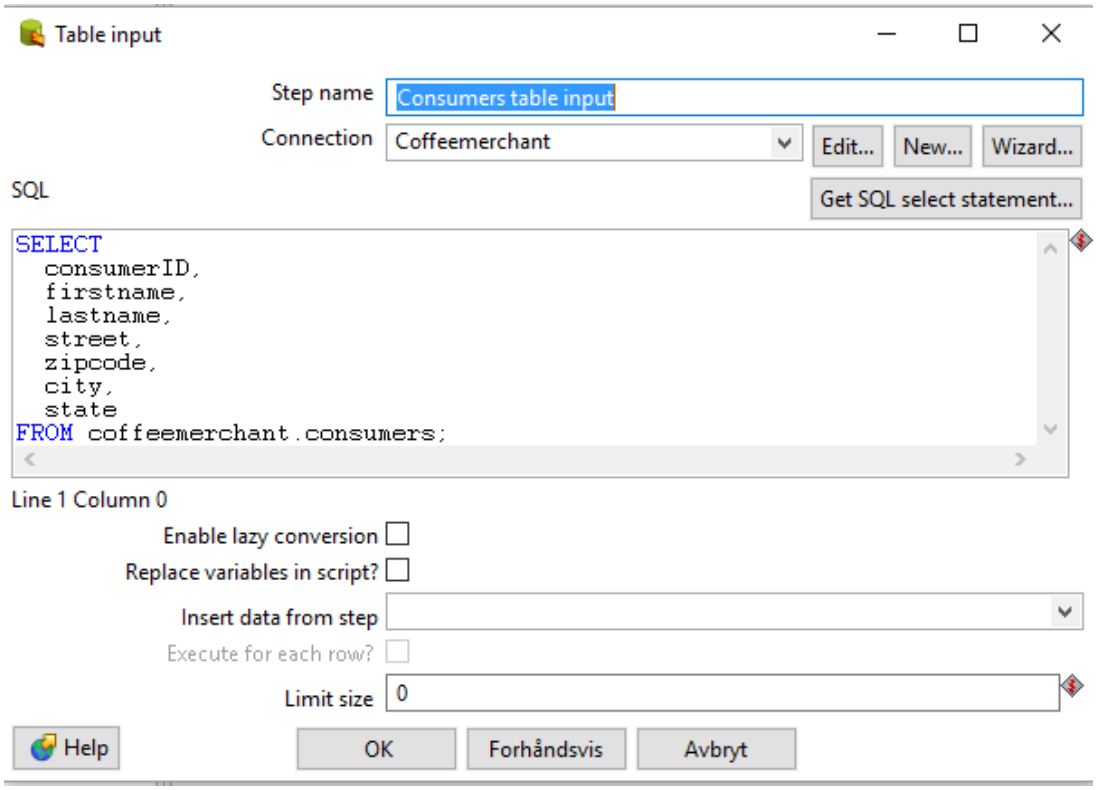


Her er det varetabellen som er valgt. Klikker vi på forhåndsvisning, får vi se at det virker:



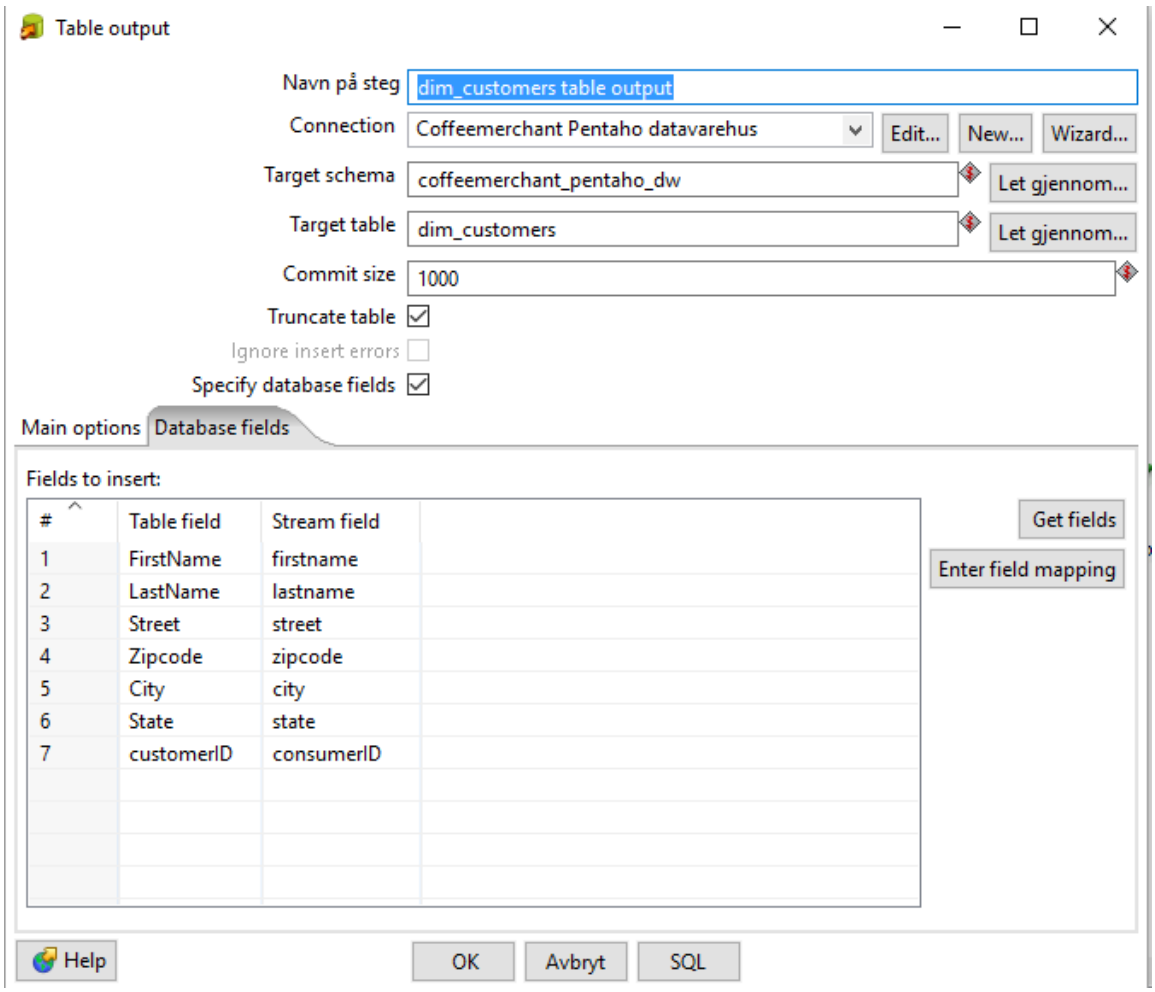
Dette ble det ikke mye modell av. Vi skal nå ta det litt videre, og vise hvordan vi kan fylle en tabell ved å bruke et input Table-steg og et output table steg. Her har jeg gjort en liten forandring i selve tabellen. Jeg har tatt bort feltet for surrogatnøkkel i dimensjonene. Da blir det litt enklere. Jeg skal her vise hvordan vi kan fylle kundedimensjonen ved å lage en enkel modell.

Steget Consumers table input er definer slik:



Her er det definert en Connection til databasen coffeemerchant.

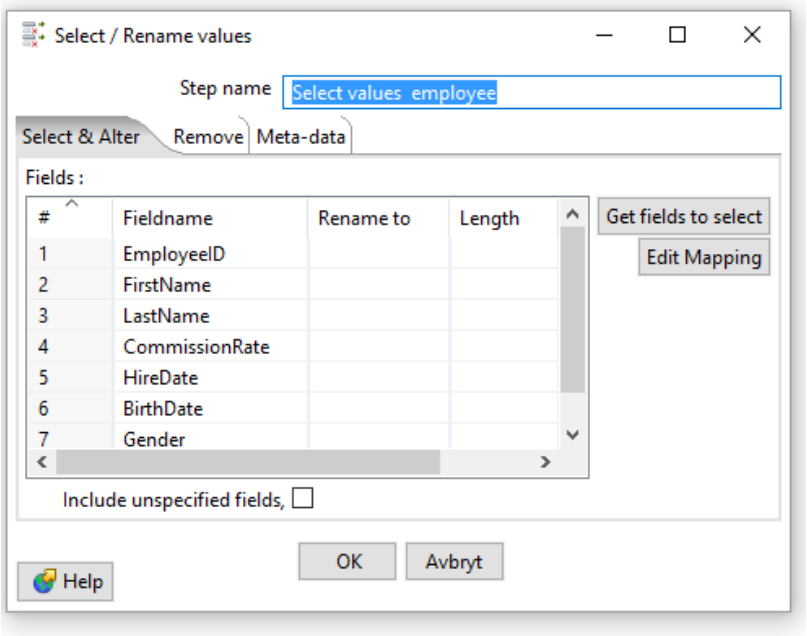
Steget dim_customers table output er definer slik:



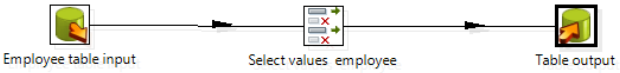
Dette er **etter** at vi har trukket en forbindelse mellom stegene (et «hopp»). Vi må også passe på å si fra at feil skal håndteres. Valget *Error handling* finner vi ved å høyreklikke på steget.

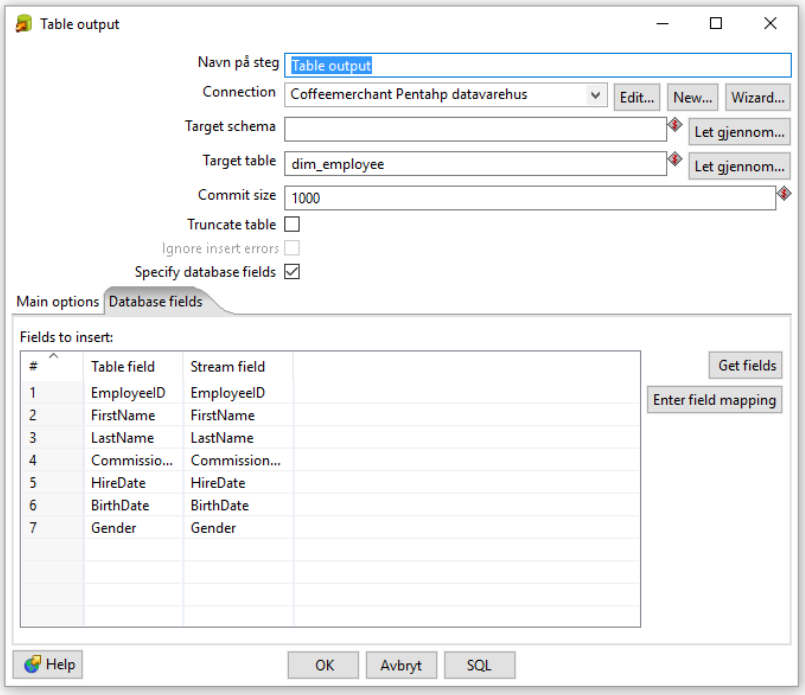
Fortsatt har vi skrevet SQL-kode. Meningen med verktøy som PDI er at vi skal kunne fylle opp datavarehuset uten å skrive SQL. Dvs., vi må skrive SELECT * FROM tabellnavn for å lese inn tabeller. Deretter velger vi ut hvilke felt vi skal ha med, for deretter å laste dem inn i datavarehustabellene.

Vi skal se på dette med ansatt-dimensjonen. Igjen har jeg fjernet surrogatnøkkellen. Vi velger et input table-steg, som bare inneholder en SELECT * FROM consumers (med connection til coffeemerchant). Deretter velger vi Select values fra mappen Transformations, og lager et hopp mellom dem. I select values-steget sletter vi bare de feltene vi ikke skal ha med (etter å ha hentet alle feltene). Dette ser nå slik ut:



Så legger vi til et Table output-steg, som er definert slik:





For både Select values og Table output krysser vi av for feilhåndtering. Når vi nå kjører modellen, ser vi at det gikk bra:

Execution Results

Execution History | Logging | Step Metrics | Performance Graph | Metrics | Preview data

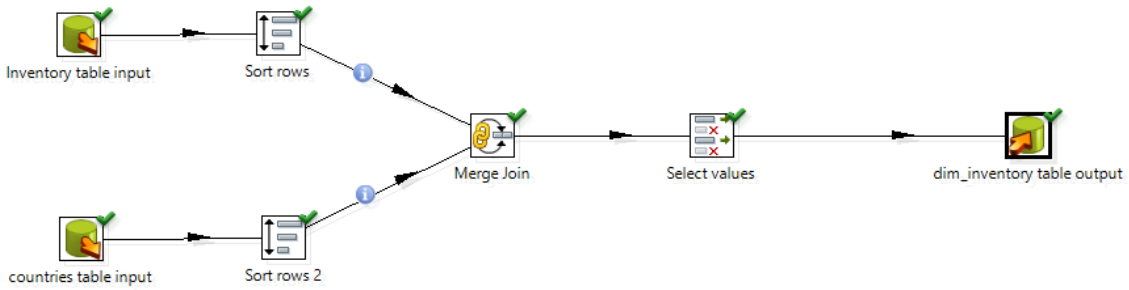
#	Stepname	Copynr	Read	Written	Input	Output	Updated	Rejected	Errors	Active	Time	Speed (r/s)	input/output
1	Employee table input	0	0	22	22	0	0	0	0	Finished	0.0s	1 222	-
2	Select values employee	0	22	22	0	0	0	0	0	Finished	0.0s	1 048	-
3	Table output	0	22	22	0	22	0	0	0	Finished	0.2s	96	-

Sjekker vi i MySql, ser vi også at dimensjonen er fylt opp. Og merk: det eneste vi har gjort i SQL er p skrive SELECT * FROM consumers!

13.4.4 Fylle varedimensjonen ved hjelp av JOIN

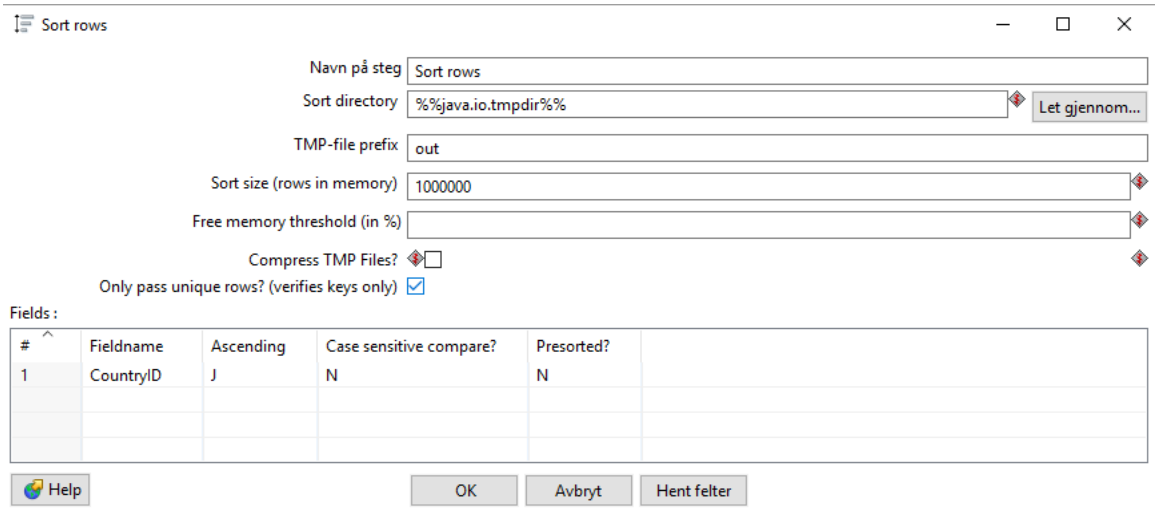
I varedimensjonen dim_inventory ønsker jeg å ha feltene inventoryID, name, price, itenType og countryname. Siden countryname ligger i tabellen countries, må vi foreta en JOIN for deretter p ta bort de attributtene vi ikke skal ha med. Primærnøkkel for countries er countryID, som også er en fremmednøkkel i inventory.

Den ferdige transformasjonen ser slik ut:



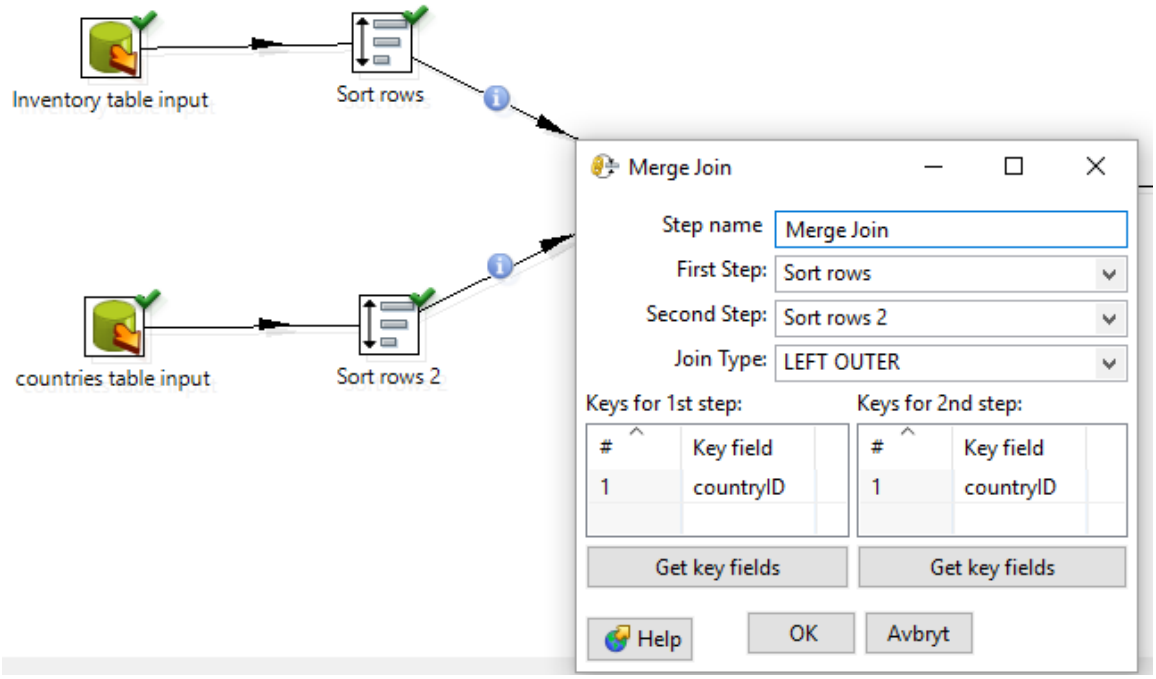
Merge Join er det steget vi skal bruke (ligger i mappen Joins). Før vi kan foreta joinen, må begge tabellene sorteres på det feltet som er grunnlag for joinen. Sort rows hentes fra mappen Transform. Vi kan hente alle feltene fra input-steget og så slette alle unntatt countryID (de slettes ikke fra datastrømmen, bare fra det det skal sorteres etter).

Definisjonen for sortering av inventory-input ser nå slik ut:



Det samme gjør vi for input fra coutry-tabellen.

Nå definerer vi selve JOIN'en:



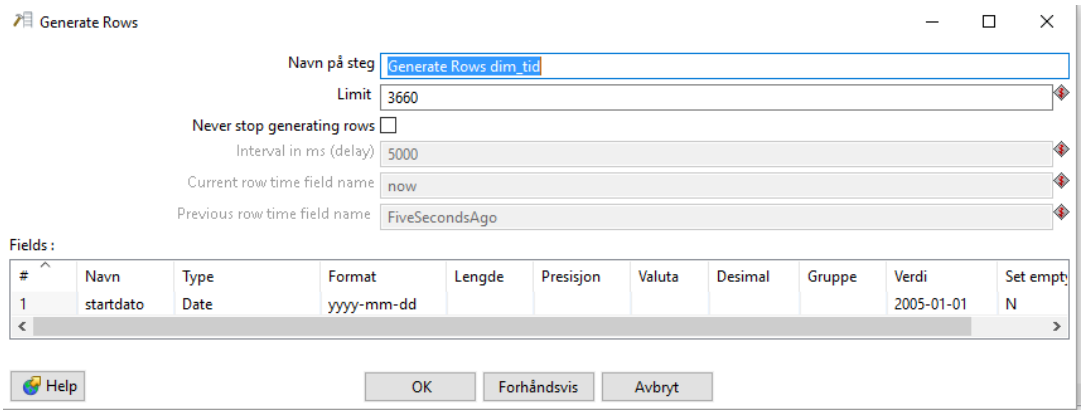
Det siste steget er å definere output slik vi har gjort for andre dimensjonstabeller. Når vi nå kjører transformasjonen, går alt som det skal. Et lite kontrollspørsmål: hvorfor har jeg spesifisert en LEFT OUTER JOIN her?

13.4.5 Fylle tidsdimensjonen

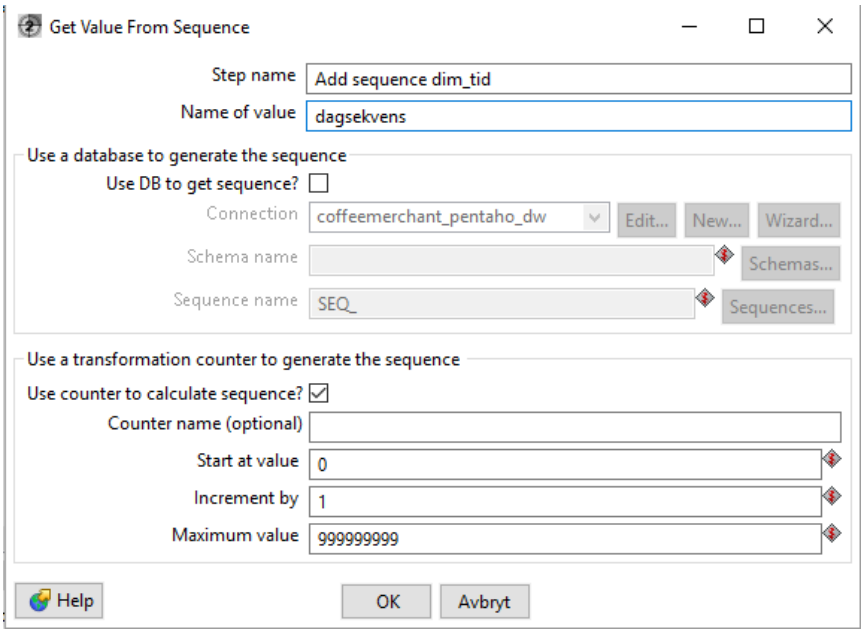
Vi skal nå bruke Spoon til å fylle tidsdimensjonen, i stedet for å kjøre en lagret prosedyre. Den ferdige transformasjonen består av fire steg, som vist nedenfor:

Vi begynner med å generere en tabell med det antall datoer vi trenger. La oss si at vi skal ha 10 år med datoer, med start i 2005. Vi lager da et felt startdato med verdien 2005-01-01 i steget Generate Rows dim_tid. Deretter lager vi en sekvens med tall. Det tredje steget er å lage datoer ved å legge sammen startdato og hvert tall i sekvensen.

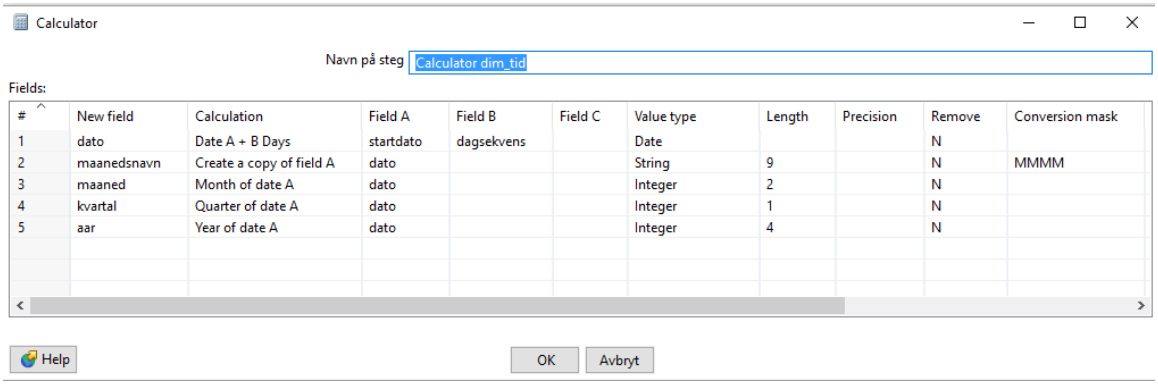
Vi begynner med å se på genereringen av datoer. Vi finner Generate Rows i Input:



Det neste er sekvensen:



Så setter vi opp de forskjellige beregningene i kalkulatoren:



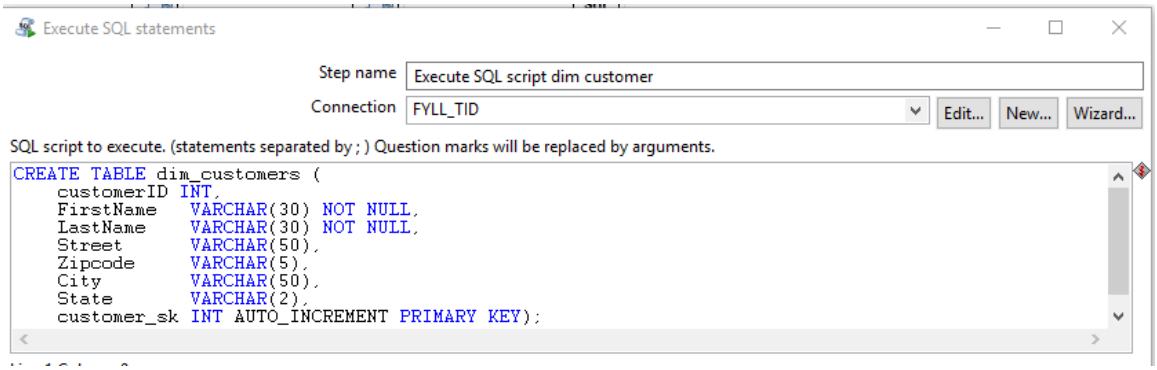
Det siste steget er output til tabellen dim_tid. Her må vi først fjerne startdato og dagsekvens fra listen over felt. Vi kjører transformasjonen, og tidsdimensjonen er fylt opp!

dato	maanedsnavn	maaned	kvartal	aar
2005-01-01	januar	1	1	2005
2005-01-02	januar	1	1	2005
2005-01-03	januar	1	1	2005
2005-01-04	januar	1	1	2005
2005-01-05	januar	1	1	2005

13.4.6 Legge inn surrogatnøkler

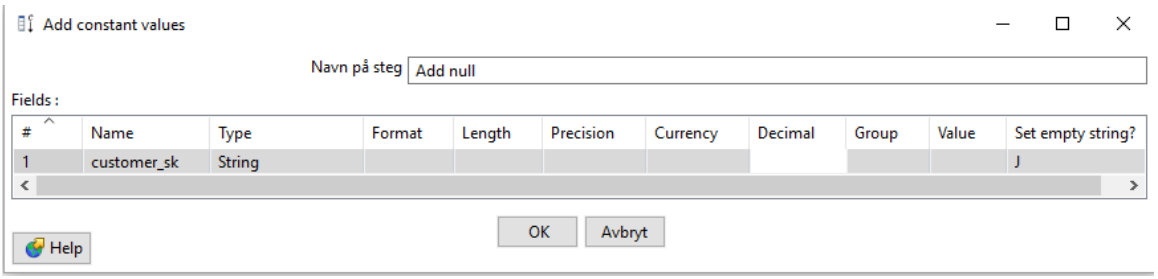
Det anbefales å alltid bruke surrogatnøkler, og det kan vi også gjøre når vi bruker Spoon. Vi skal her se på hvordan det gjøres for kundedimensjonen.

Først oppretter vi dimensjonstabellen, med surrogatnøkkelen sist i listen over felt:

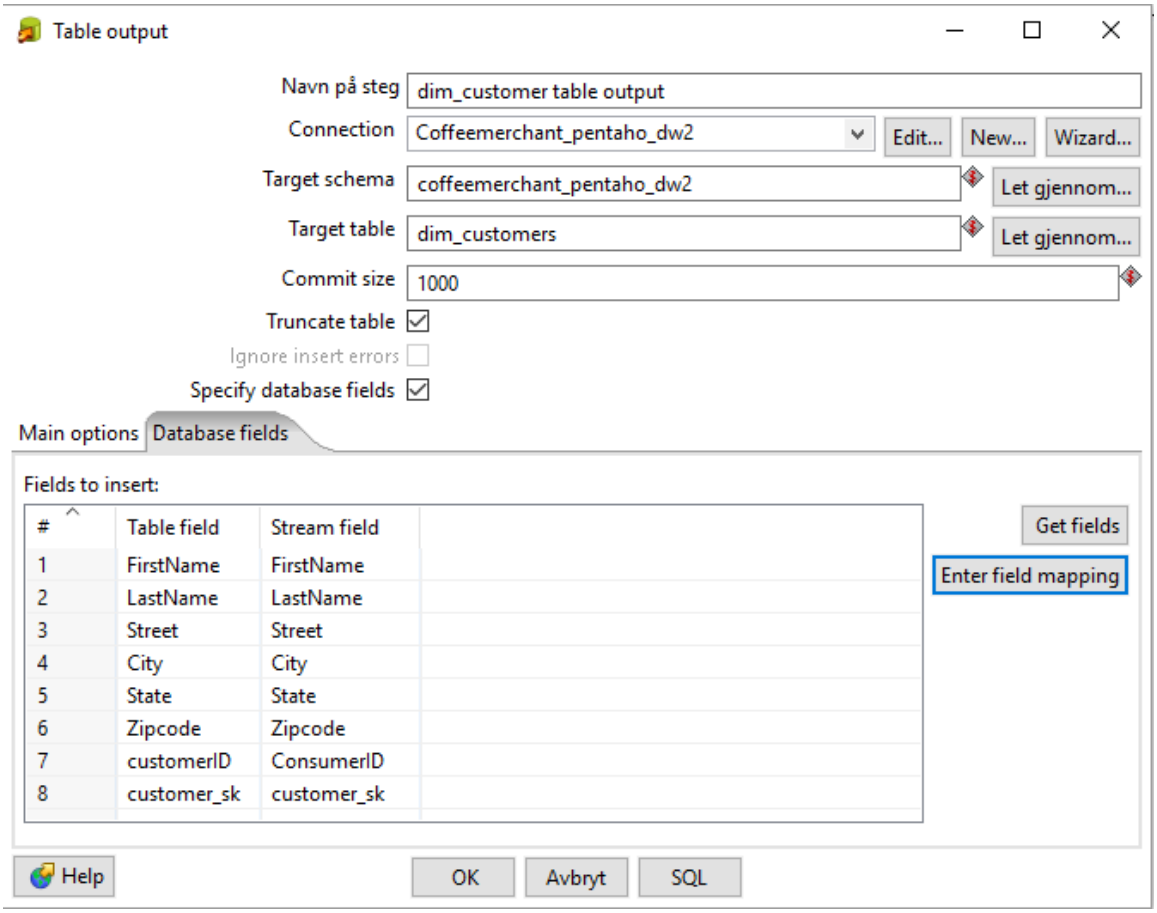


Så skal vi lage en ny transformasjon som fyller dimensjonstabellen. Den ferdige transaksjonen ser slik ut:

Det nye her er steget kalt Add null. Dette er nødvendig, siden vi vet at et felt som er definert med AUTO_INCREMENT må ha null som innsettingsverdi. Steget ser slik ut:



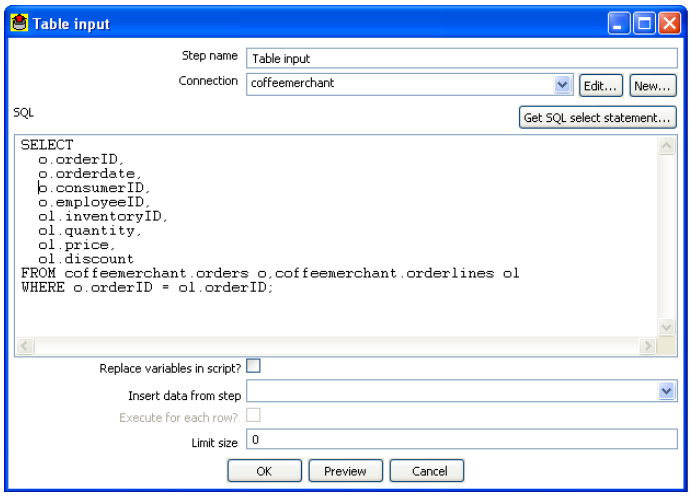
Her er det definert et felt customer_sk. Legg merke til at det står en «J» under overskriften «**Set empty string?».** Nå er det bare å mappe dette feltet i tillegg til de som mottas fra «Select values»-steget:



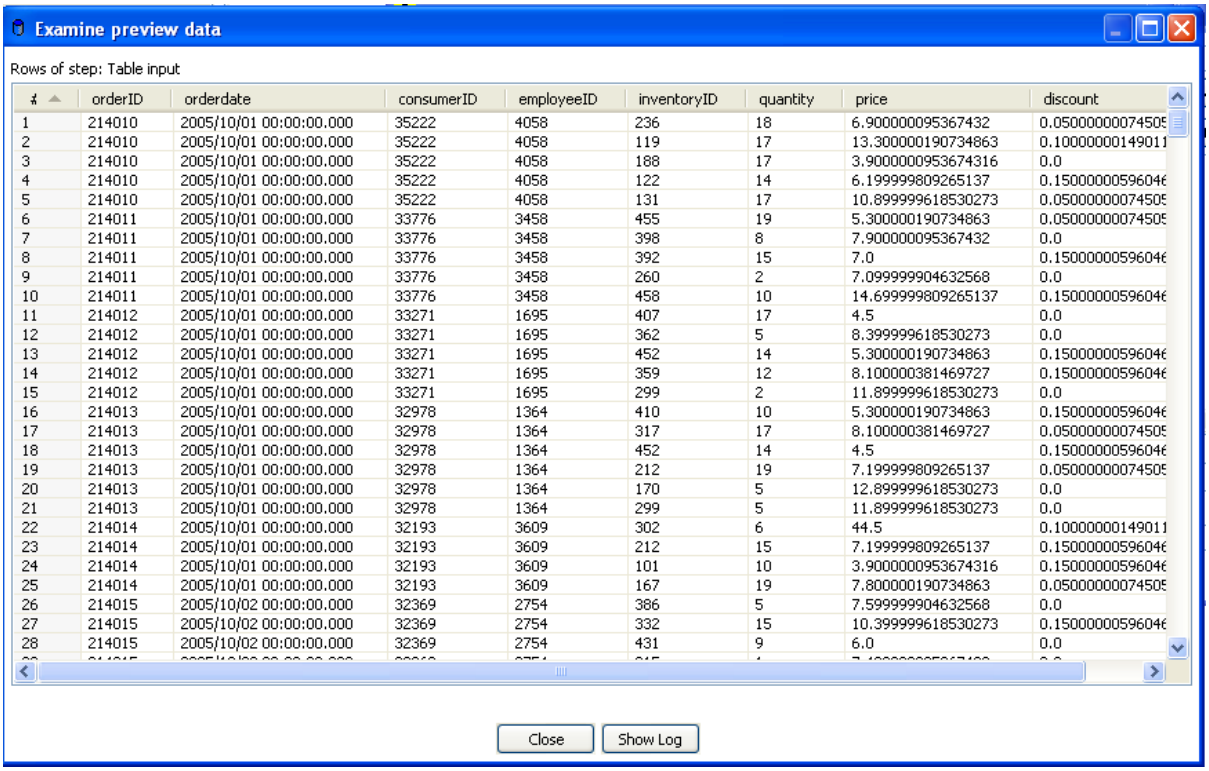
13.4.6 Lage transformasjon for faktatabellen med stagingtabeller

En faktatabell vil ofte være resultatet av en JOIN. I vårt eksempel en JOIN mellom tabellene orders og orderlines. I tillegg skal vi, som tidligere, hente inn surrogatnøkler fra dimensjonstabellene (noe som betyr at disse må være fylt opp før vi kan fylle faktatabellen). Vi skal gjøre dette med et steg for hver surrogatnøkkel. I MySQL lager vi ferdig fire stagingtabeller for dette formålet. Vi kan opprette dem over samme lest, det vil si uten definerte surrogatnøkler. Spoon vil legge til de nye feltene etter hvert.

Når vi skal ha en JOIN av to tabeller i samme database, anbefaler Pentaho å gjøre dette direkte i SQL. Vi drar et input-ikon ut på arbeidsflaten og dobbeltklikker. Deretter skriver vi inn SELECT-setningen direkte:



Ved å trykke på **Preview** kan vi se at dette gikk bra:



#	orderID	orderdate	consumerID	employeeID	inventoryID	quantity	price	discount
1	214010	2005/10/01 00:00:00.000	35222	4058	236	18	6.900000095367432	0.05000000074505
2	214010	2005/10/01 00:00:00.000	35222	4058	119	17	13.300000190734863	0.10000000149011
3	214010	2005/10/01 00:00:00.000	35222	4058	188	17	3.9000000953674316	0.0
4	214010	2005/10/01 00:00:00.000	35222	4058	122	14	6.199999809265137	0.15000000596046
5	214010	2005/10/01 00:00:00.000	35222	4058	131	17	10.899999618530273	0.05000000074505
6	214011	2005/10/01 00:00:00.000	33776	3458	455	19	5.300000190734863	0.05000000074505
7	214011	2005/10/01 00:00:00.000	33776	3458	398	8	7.900000095367432	0.0
8	214011	2005/10/01 00:00:00.000	33776	3458	392	15	7.0	0.15000000596046
9	214011	2005/10/01 00:00:00.000	33776	3458	260	2	7.099999904632568	0.0
10	214011	2005/10/01 00:00:00.000	33776	3458	458	10	14.699999809265137	0.15000000596046
11	214012	2005/10/01 00:00:00.000	33271	1695	407	17	4.5	0.0
12	214012	2005/10/01 00:00:00.000	33271	1695	362	5	8.399999618530273	0.0
13	214012	2005/10/01 00:00:00.000	33271	1695	452	14	5.300000190734863	0.15000000596046
14	214012	2005/10/01 00:00:00.000	33271	1695	359	12	8.100000381469727	0.15000000596046
15	214012	2005/10/01 00:00:00.000	33271	1695	299	2	11.899999618530273	0.0
16	214013	2005/10/01 00:00:00.000	32978	1364	410	10	5.300000190734863	0.15000000596046
17	214013	2005/10/01 00:00:00.000	32978	1364	317	17	8.100000381469727	0.05000000074505
18	214013	2005/10/01 00:00:00.000	32978	1364	452	14	4.5	0.15000000596046
19	214013	2005/10/01 00:00:00.000	32978	1364	212	19	7.199999809265137	0.05000000074505
20	214013	2005/10/01 00:00:00.000	32978	1364	170	5	12.899999618530273	0.0
21	214013	2005/10/01 00:00:00.000	32978	1364	299	5	11.899999618530273	0.0
22	214014	2005/10/01 00:00:00.000	32193	3609	302	6	44.5	0.10000000149011
23	214014	2005/10/01 00:00:00.000	32193	3609	212	15	7.199999809265137	0.15000000596046
24	214014	2005/10/01 00:00:00.000	32193	3609	101	10	3.9000000953674316	0.15000000596046
25	214014	2005/10/01 00:00:00.000	32193	3609	167	19	7.800000190734863	0.05000000074505
26	214015	2005/10/02 00:00:00.000	32369	2754	386	5	7.599999904632568	0.0
27	214015	2005/10/02 00:00:00.000	32369	2754	332	15	10.399999618530273	0.15000000596046
28	214015	2005/10/02 00:00:00.000	32369	2754	431	9	6.0	0.0

Siden vi nå bare har selektert de feltene vi skal ha med, trenger vi ikke noe Select values steg. Vi skal nå gå videre med å legge inn surrogatnøklene. Vi henter et Table lookup-ikon og angir hvordan lookup skal skje:

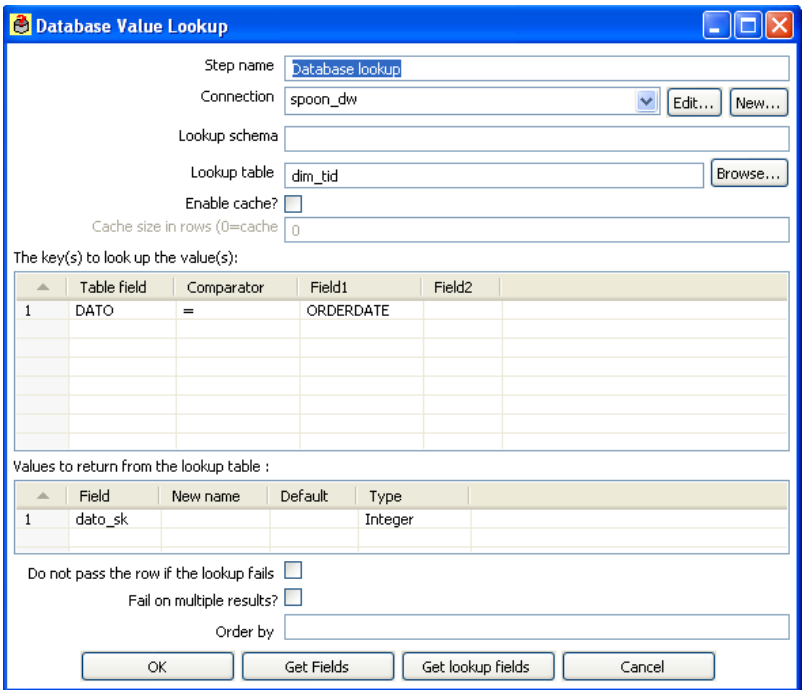
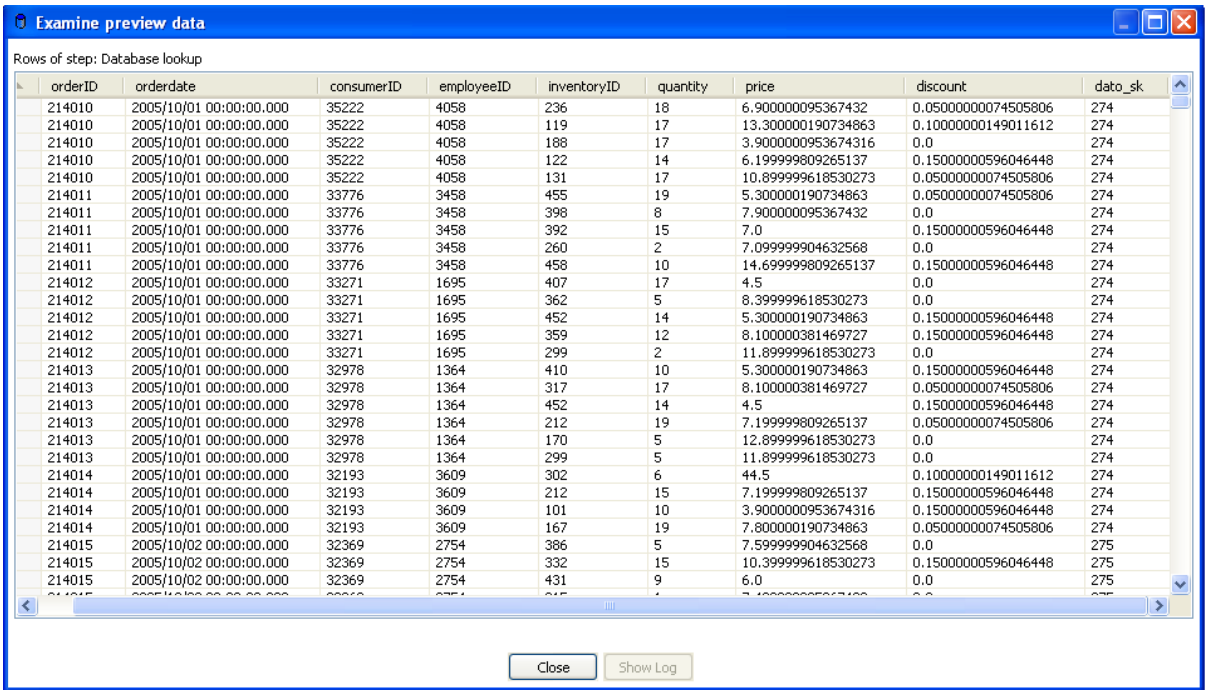
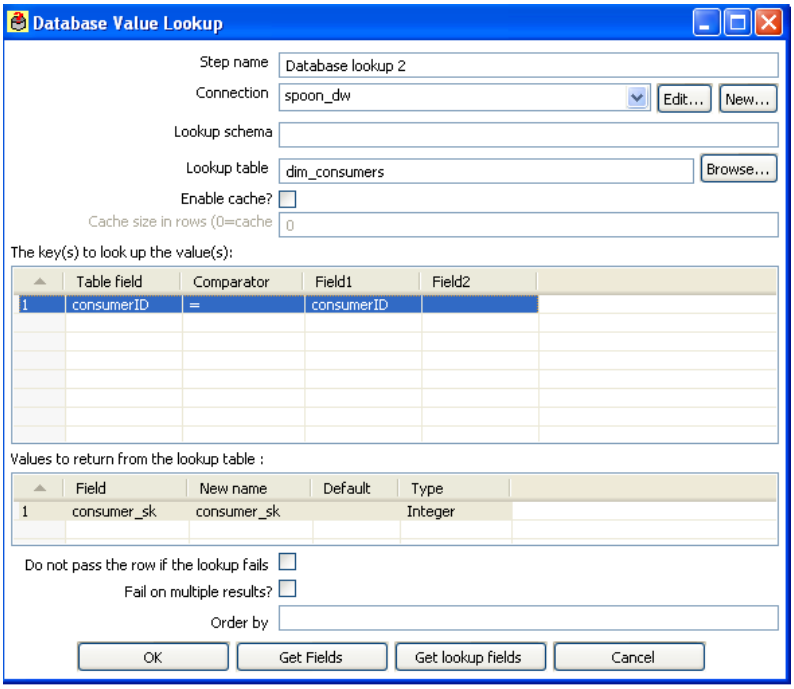


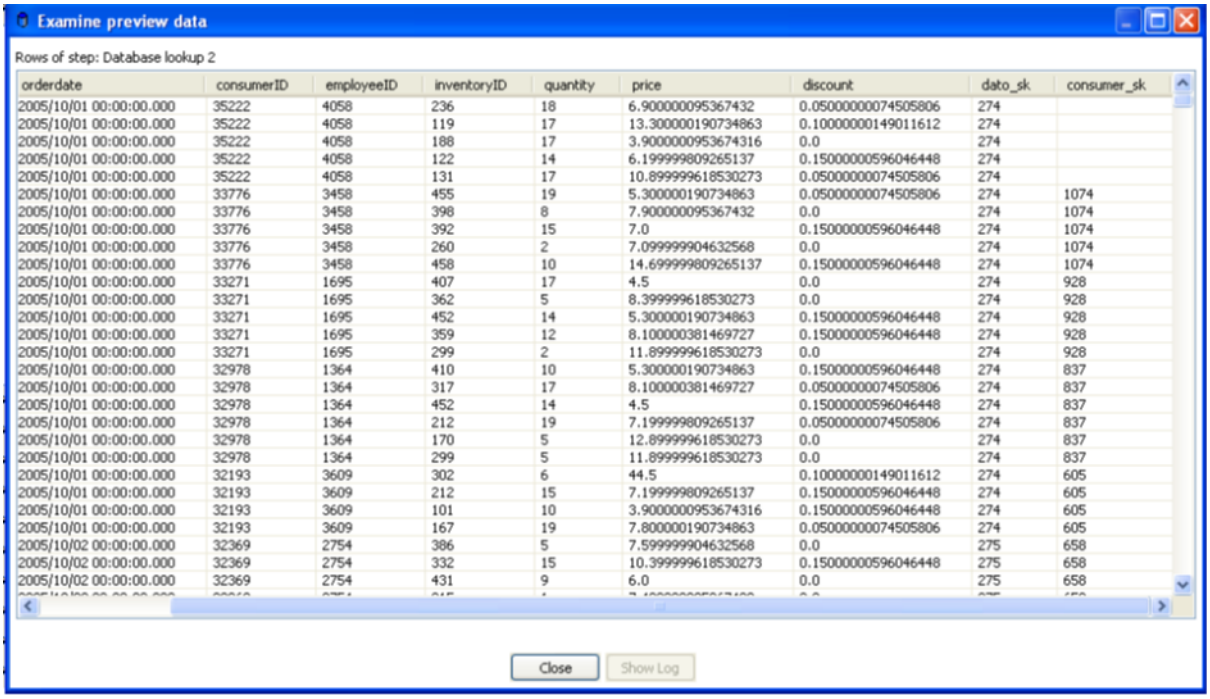
Table field er feltet i lookup-tabellen (dim_tid), mens Field1 er i spørringen vi har kjørt mot kilden. Vi kjører et preview av steget, og ser at dato_sk er lagt til i tabellen med verdier:



Vi fortsetter med å hente ut surrogatnøkler for kundene. Først introduserer vi en ny stagingtabell, deretter lager vi en ny lookup:



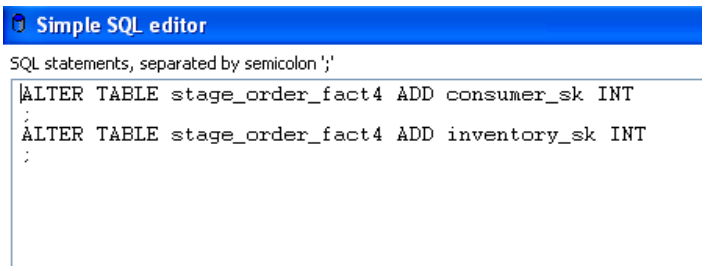
Igjen kjører vi en preview på dette:



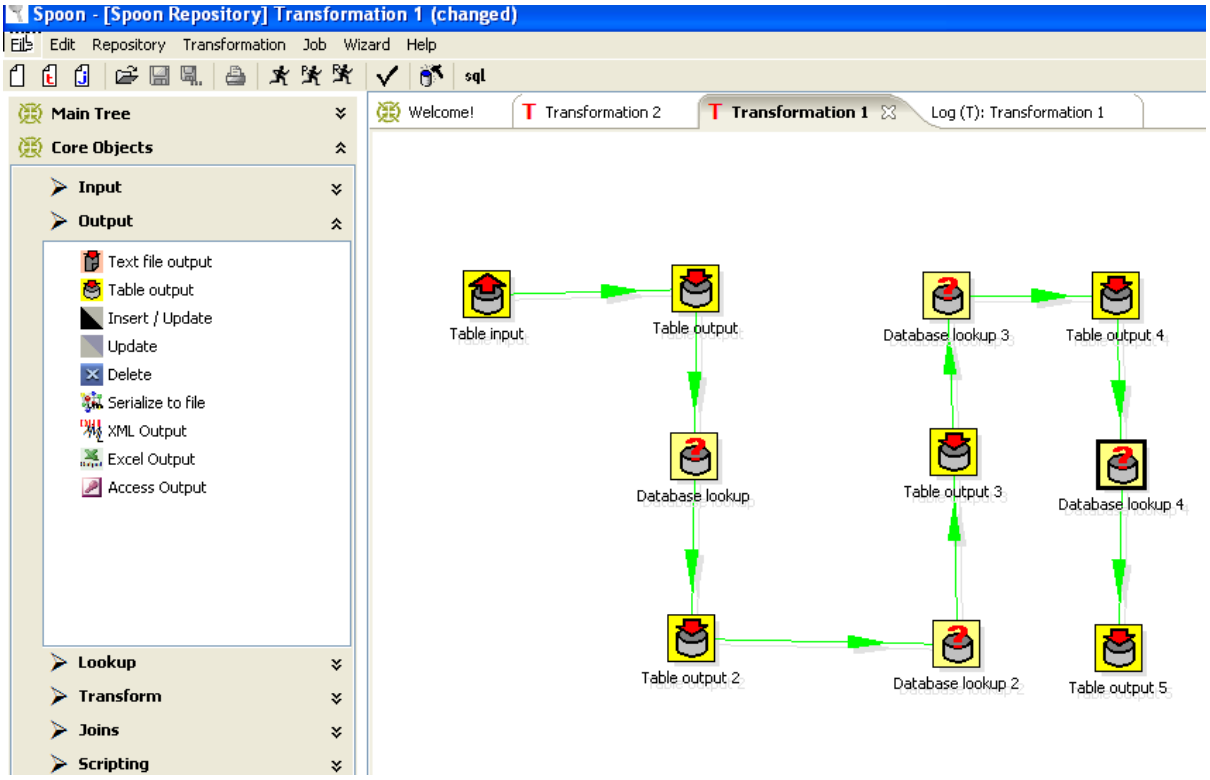
Vi ser at noen av kundene mangler en surrogatnøkkel. Dette skal vi ta oss av senere.

Vi fortsetter på samme måte med å lage oppslag på surrogatnøkler for ansatt og vare.

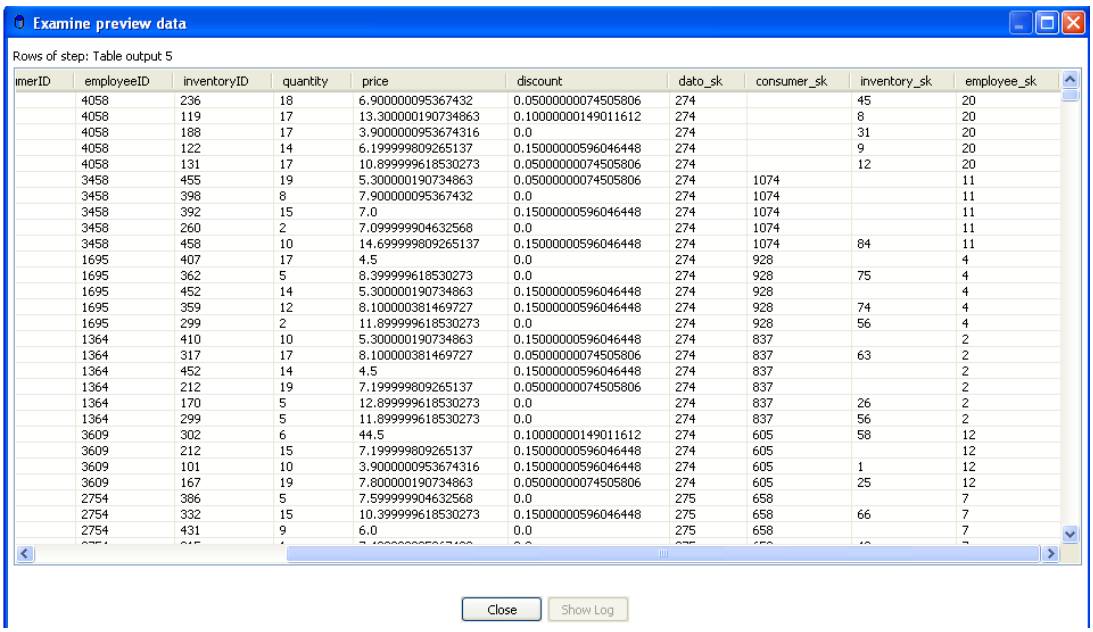
Hvis de nye surrogatnøkkelfeltene ikke finnes i stagingtabellene (som når vi bare har kopiert den første av dem), må vi trykke på SQL-knappen i definisjonsvinduet for output-tabellen og velge Execute (nederst i SQL-vinduet). SQL-koden for stagingtabell 3 ser slik ut:



Etter det siste oppslaget legger vi data inn i faktatabellen fact_sales. Modellen vår ser nå slik ut:



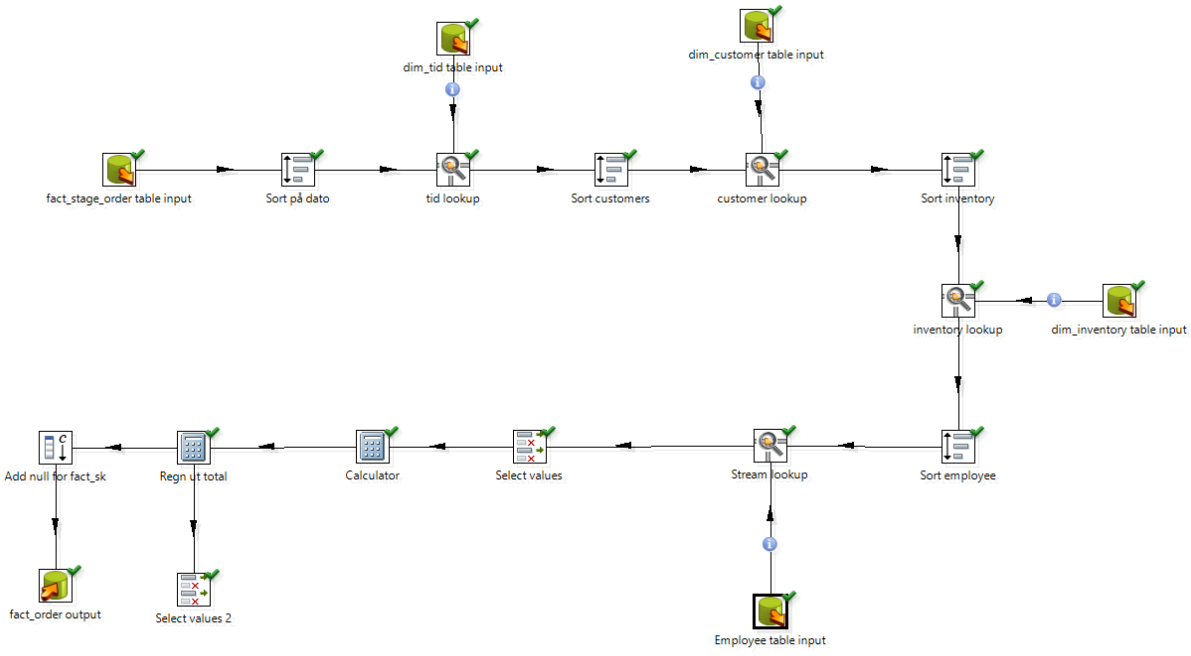
Til slutt kan vi kontrollere at alt fungerer, og vi får med samtlige surrogatnøkler. Vi kjører transformasjonen som en jobb med preview:



Vi ser at alle surrogatnøkler er pent på plass.

13.4.7 Transformasjon til faktatabell uten staging

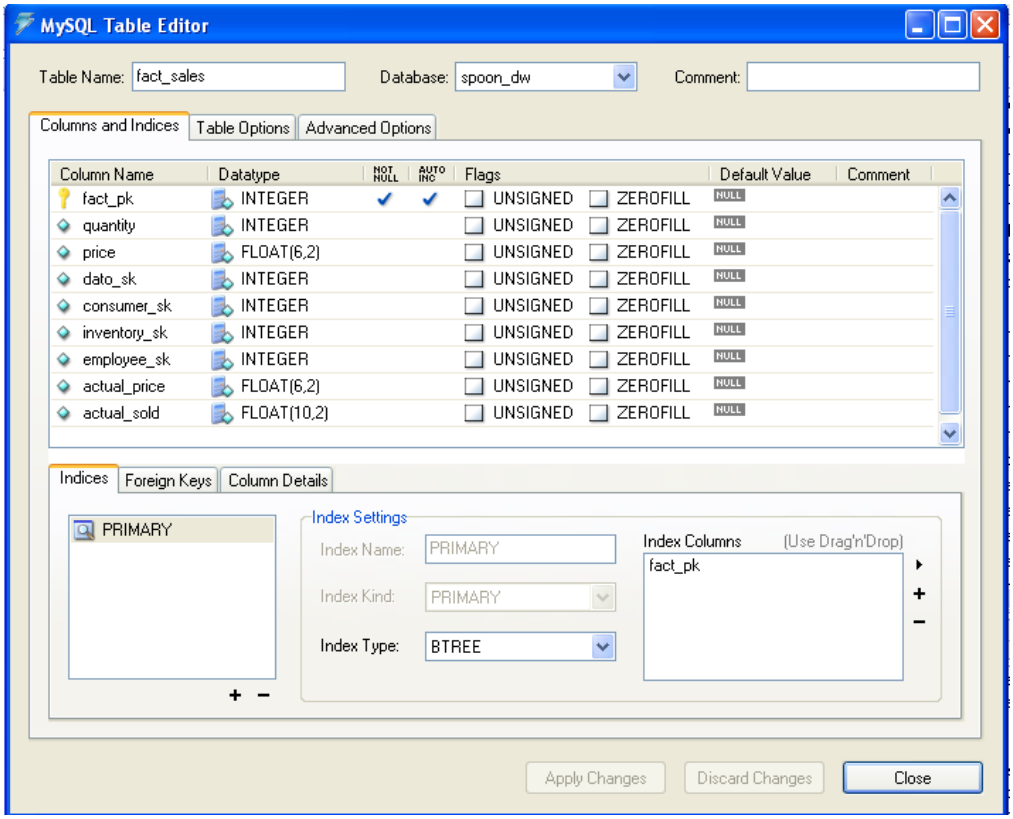
I stedet for å bruke flere stagingtabeller, kan vi oppnå det samme ved å bruke såkalt Stream Lookup. En modell av den samme transformasjonen som nettopp er vist, men uten stagingtabeller, ser slik ut:



Forklaring:

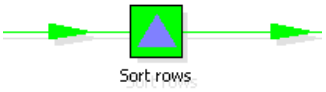
- Data leses fra samme kilder som tidligere, med samme SQL-setning for JOIN.
- Før en Stream lookup må datasettet sorteres på det eller de kolonnene som skal brukes til oppslag, her på dato
- En stream lookup henter så surrogatnøkkelen fra tidsdimensjonen.
- Ny sortering, denne gang på kunde-ID.
- Ny stream lookup i kundedimensjone gir surrogatnøkkel for kunde.
- Ny sortering, denne gang på vare-ID.
- Stream lookup på varedimensjonen gir surrogatnøkkel for vare.
- Ny sortering på ansatt-ID.
- Stream lookup på ansattdimensjonen.
- Kalkulator beregner actual_price (pris etter at rabatt er trukket fra).
- Kalkulator beregner actual_sold (actual_price*quantity).
- Filtrering: fjerner de kolonnene som ikke skal lasts over i måltabellen.
- Lasting til fact_sales.

Tabellen fact_sales har nå en litt annerledes struktur, for å passe med rekkefølgen transformasjonen gir på kolonnene:



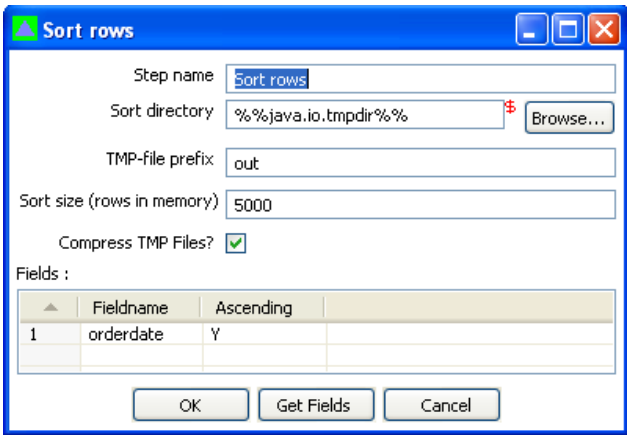
Output fra filtrering må være lik definisjonen av faktatabellen.

Vi skal nå se nærmere på de stegene som er brukt i denne transformasjonen. Vi skal nøye oss med et eksempel på hver type steg.



Sortering:

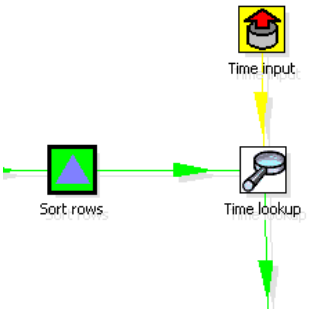
Vi skal her vise definisjonen av sortering på ordredato. Som vanlig høyreklikker vi på steget, og velger Edit:



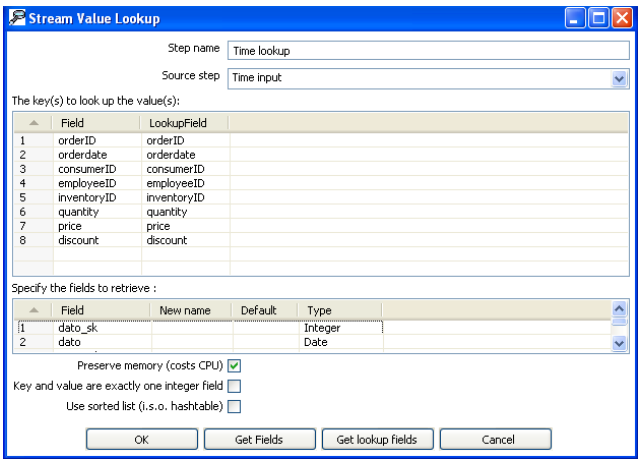
Vi kan klikke på Get Fields først, og så slette alle som ikke skal være med som grunnlag for sortering. Vi kan godt sortere på en kombinasjon av flere felt (som etternavn pluss fornavn).

Stream lookup:

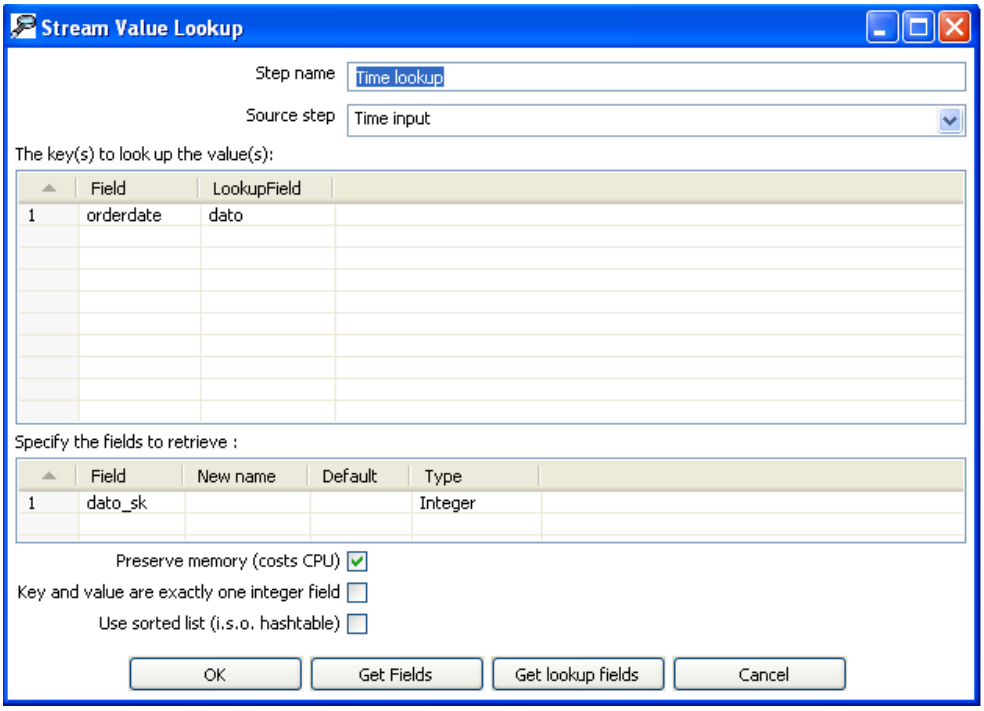
Etter sortering kan vi slå opp i tidsdimensjonen. Først må vi knytte sammen både sorteringen og dimensjonstabellen med en stream lookup. Her er lookup'en også omdøpt til Time lookup:



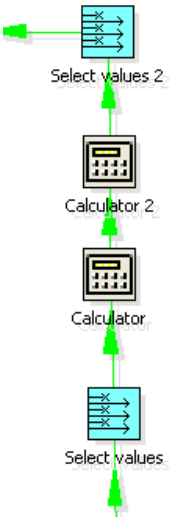
Definisjonen får vi igjen gjøre ved å velge Edit. Vi kan først hente inn alle feltene fra sorteringen og alle feltene fra oppslagstabellen. Når dette er gjort (merk knappene nederst i vinduet), ser det slik ut:



Vi sletter nå alle feltnavn som ikke skal være med, og sitter igjen med dette:

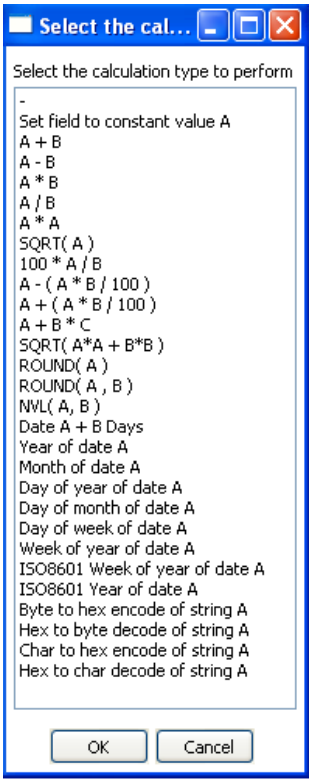


Slik fortsetter vi med å gjøre oppslag på surrogatnøkler. Til slutt er vi kommet til utregningene, som ikke var med i den første utgaven av transformeringen. I datavarehuset er det jo ikke rabatten vi skal lagre, men pris etter rabatt (actual_price). Vi skal også lagre samlet pris, altså actual_price * quantity, som actual_sold (dette gjorde vi i den opprinnelige opprettelsen av datavarehuset med SQL). Siden vi skal gjøre to utregninger, trenger vi to kalkulatorer. Kalkulator ligger under Transformations.

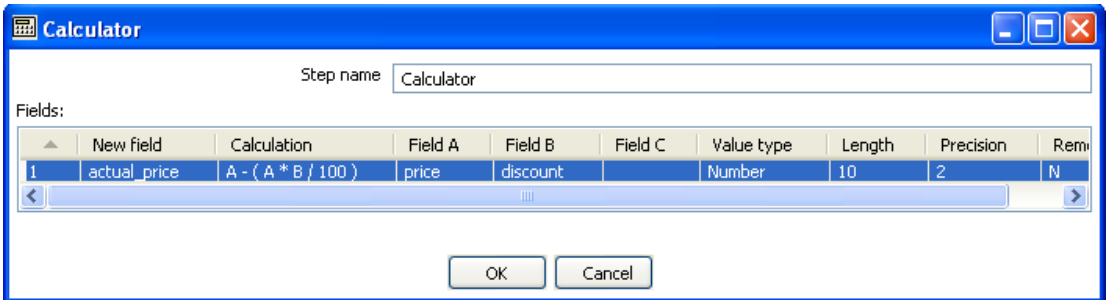


Utregninger:

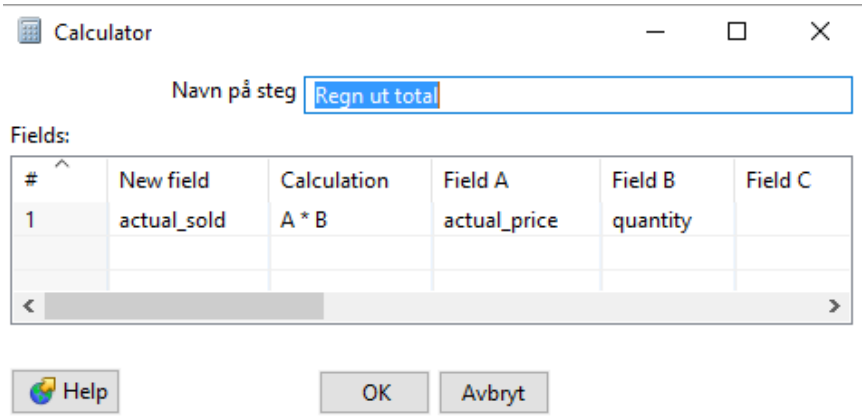
Vi åpner den første kalkulatoren for redigering. Vi må definere hvilke felt som skal være med i utregningen, og hva slags utregning som skal foretas. Her må vi velge utregning fra en meny som presenteres ved å klikke på Calculation-feltet:



Den ferdige definisjonen av den første utregningen ser slik ut:



Den andre utregningen regner ut verdien for feltet actual_sold:



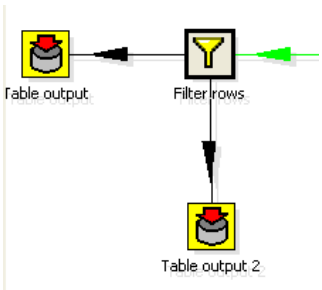
13.4.8 Håndtering av avvik

En sentral del av data staging er å håndtere feil og avvik i data. Dette kan vi definere i Spoon. Vi har tidligere sett at det finnes kundenummere i ordretabellen som ikke har noen tilsvarende postering i kundetabellen. Det samme gjelder varer. Spørsmålet nå er hva vi skal gjøre med slike avvik. Det er flere måter å håndtere dette på, som:

- Avvikende poster lastes til en egen avvikstabell, som senere kan kontrolleres manuelt
- Manglende nøkler erstattes med bestemte verdier.

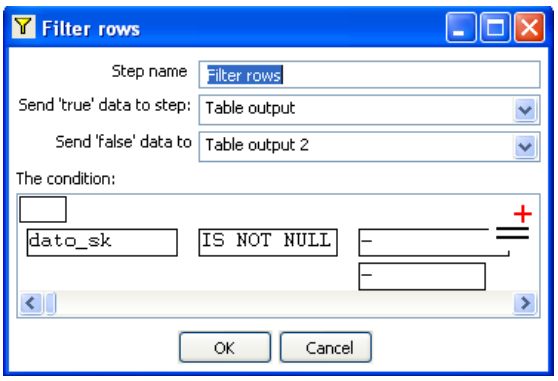
Vi skal demonstrere hvordan dette kan gjøres med transformasjonen av fakta.

Vi skal til slutt ordne opp i problemet med manglende surrogatnøkler. Det er flere måter å rydde opp i dette på. Man kan sette inn en defaultverdi, eller man kan sende poster med manglende nøkler til en "ventefil". Der kan de gås gjennom manuelt dagen etter.

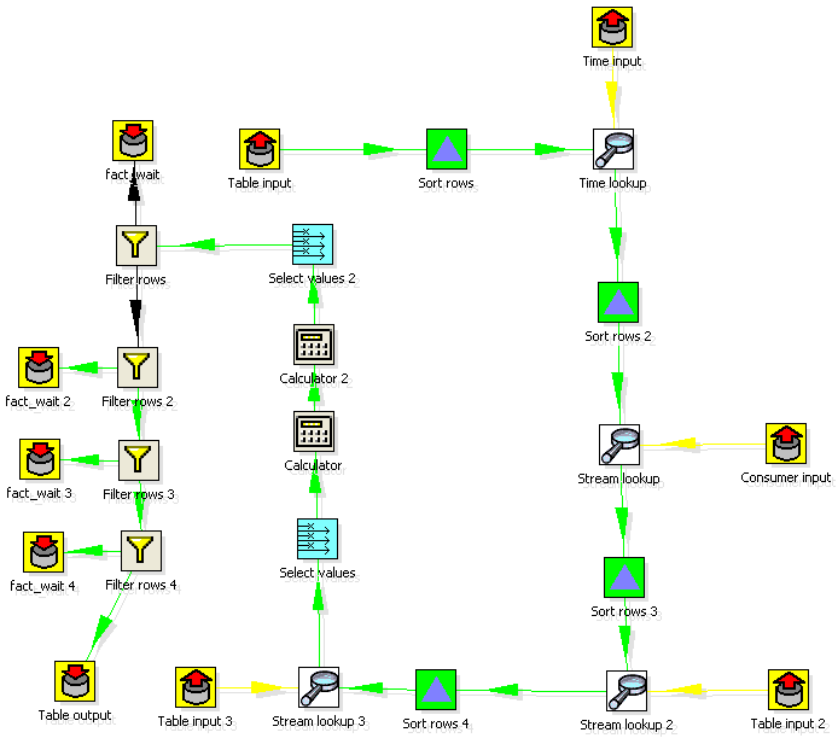


Kanskje vil man bruke en kombinasjon av disse metodene. Uansett må man skille ut de postene som har mangler, og det gjør man ved å sette inn et filter. Nedenfor er satt inn et filter for å skille ut poster med manglende tidsnøkkel:

Table output er fortsatt tabellen fact_sales. Table output 2 er en tabell kalt fact_wait, som har samme struktur som faktatabellen med unntak av at den ikke har noen egen surrogatnøkkel (tabellen fact_sales har en egen fact_sk som primærnøkkel). Nedenfor er vist definisjonen på filtreringen.



Her er bare vist for behandling av manglende surrogatnøkkel for tid. Vi må lage egne filtere også for kunder, varer og ansatte. Den endelige modellen vil da se slik ut:

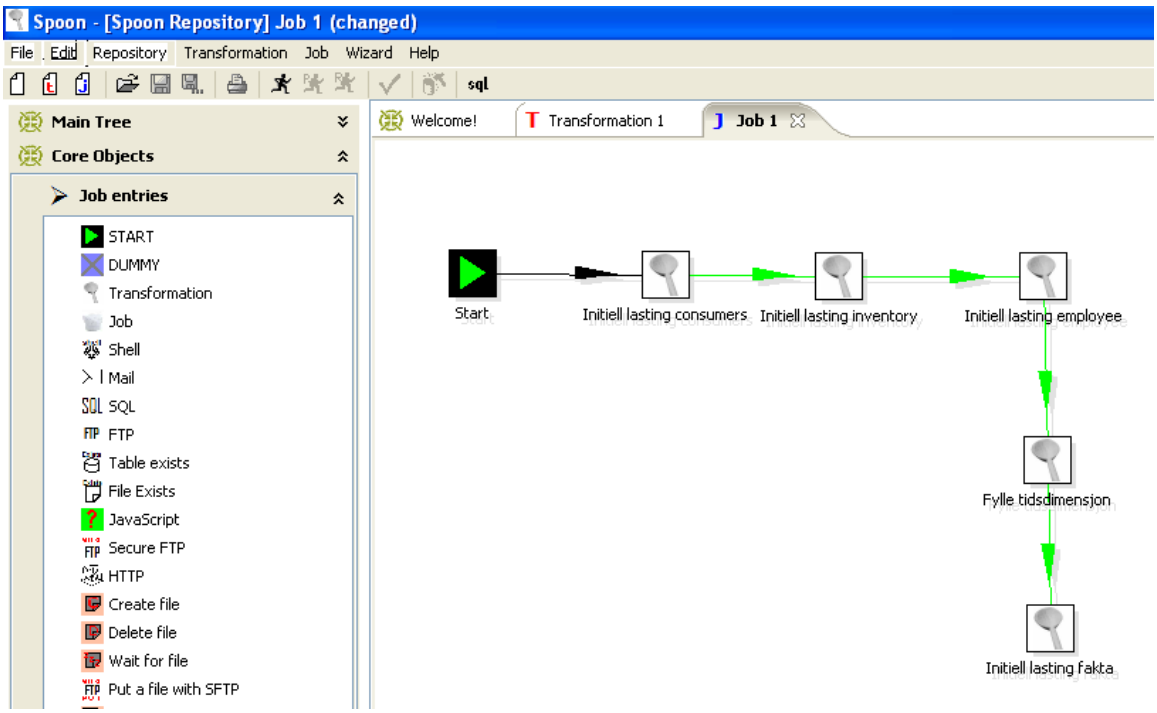


13.4.8 Definere jobber

Når vi er ferdige med å modellere transformasjonene, sitter vi igjen med en mengde transformasjoner. Vi må nå definere jobber, som knytter transformasjoner sammen i en prosessflyt. De transformasjonene vi har vist hittil, er alle for den initielle oppfyllingen av datavarehuset med eksisterende data. Vi kan beskrive prosessflyten slik:

- Last kundedata til kundedimensjonen.
- Last varedata til varedimensjonen.
- Last ansattdata til ansattdimensjonen.
- Fyll tidsdimensjonen.
- Last salgsfakta fra ordre- og ordrelinjetabellene.

Nedenfor er vist en meget enkel jobbdefinisjon av dette.

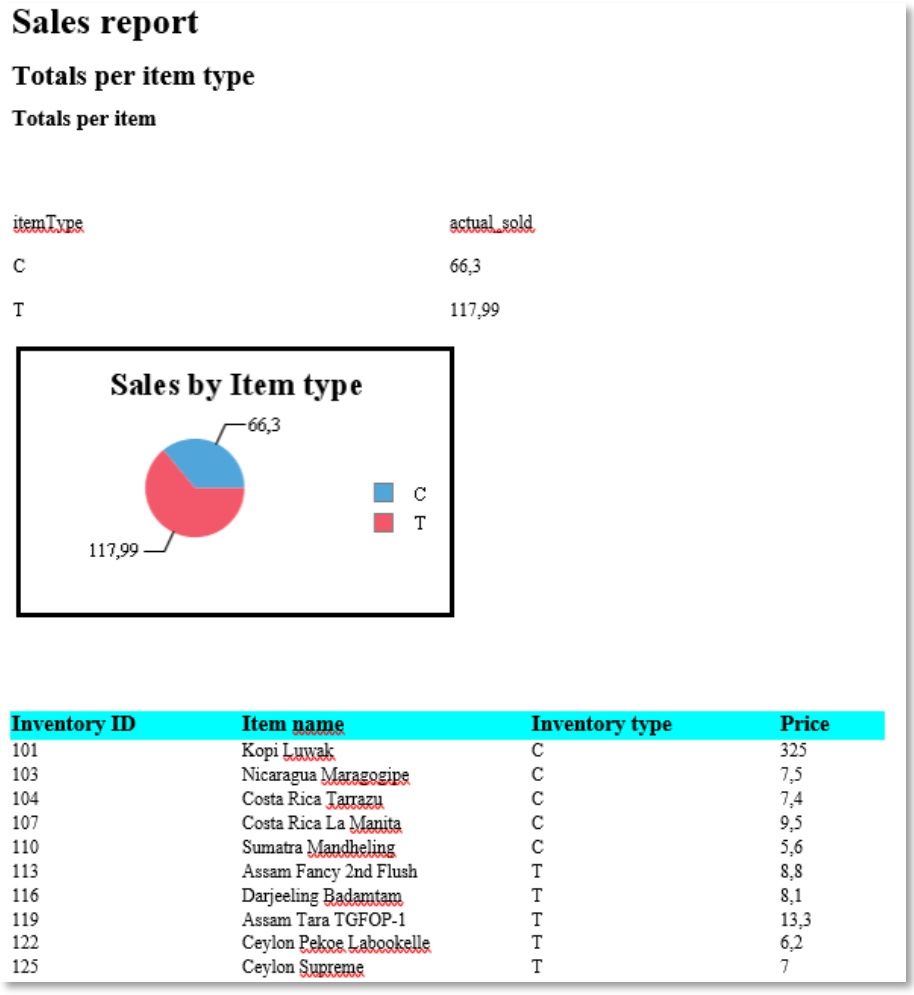


Kapittel 14 Visualisering av data

Data i databaser er nettopp det: data. Det vi trenger for å ha nytte av dem, er at de kan gjøres om til informasjon. Måten vi gjør dette på, er ved visualisering. Vi har tidligere brukt SQL-spørringer for å presentere data. Dette er imidlertid ikke noe for sluttbrukerne, altså de som skal ha informasjonen. I de neste kapitlene skal vi se hvordan vi kan visualisere data ved hjelp av verktøy som (iallfall i prinsippet) sluttbrukerne selv kan benytte.

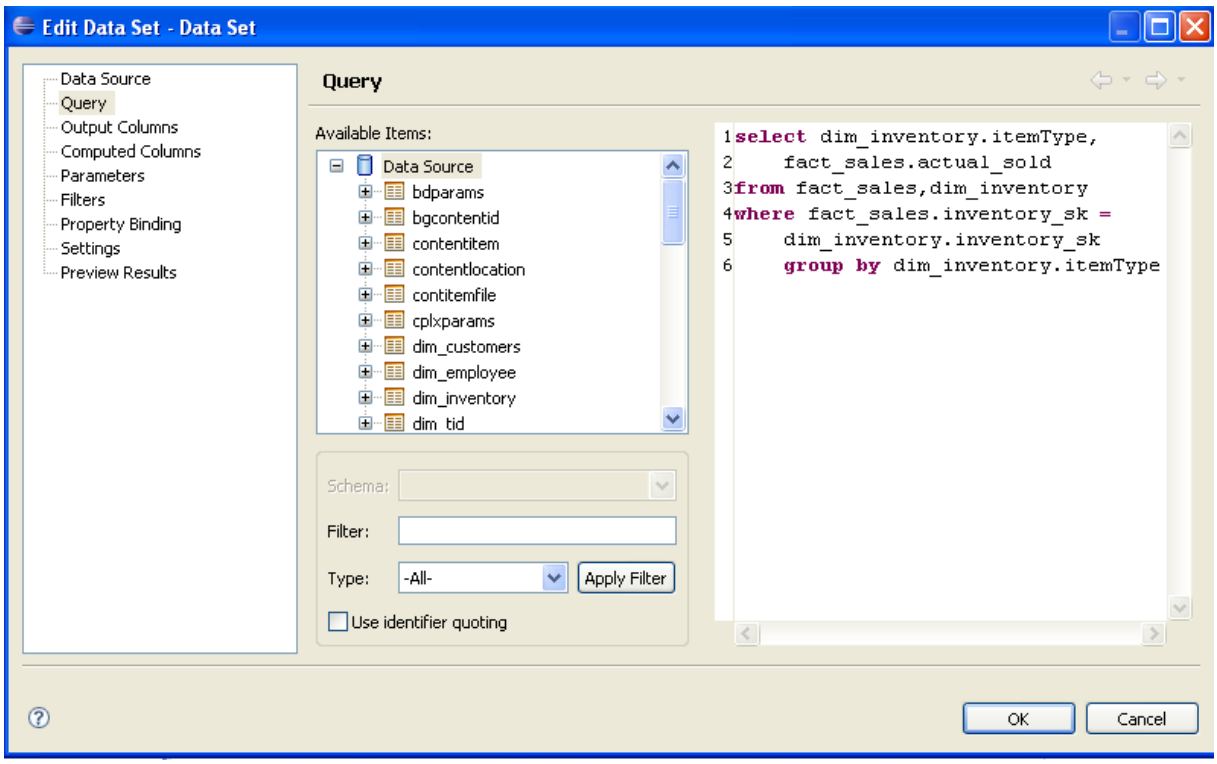
14.1 Rapporter

Data på transaksjonsnivå gir i praksis ingen informasjon. De må forenkles før de kan tolkes. En klassisk måte å presentere data på, er som rapporter. Hvis data ligger i en relasjonsdatabase, vil det bak rapporten ligge en SQL-spørring. Nedenfor er vist en ganske enkel rapport laget med en Eclipse plug-in kalt BIRT (Business Intelligence Report Tool):



Figur 14.1 Enkel rapport laget med en rapportgenerator (BIRT)

I BIRT lages en SELECT-setning for å lese data fra databasen. Her er vist setningen som ligger til grunn for rapporten over:



Figur 14.2 SELECT-setning som gjør det mulig å lage rapporten i figur 14.1

Så langt er ikke dette noe sluttbrukerverktøy. Når data først er lest inn, kan imidlertid sluttbrukerne selv designe rapporten med tabeller, grafikk, overskrifter, forklaringer m.m.

Et enklere, men samtidig mer avansert verktøy er Microsoft Power BI, som vi skal bruke i dette kurset. Power BI brukes til selve presentasjonen av data. For å lage en rapport med tekst, overskrifter osv. bruker vi en tekstbehandler, som vi så kopierer presentasjonene inn i.

I tillegg til Power BI skal vi også bruke regnearket Microsoft Excel.

Nedenfor er vist en visualisering laget med Power BI:

countryname	C	T	Total
	39 672,17	20 497,18	**60 169,35**
Brazil	2 812,39		**2 812,39**
China		25 393,75	**25 393,75**
Colombia	2 541,85		**2 541,85**
Costa Rica	2 670,14		**2 670,14**
Egypt		4 424,59	**4 424,59**
Ethiopia	4 382,74		**4 382,74**
Germany		4 169,90	**4 169,90**
Guatemala	1 783,49		**1 783,49**
Haiti	567,20		**567,20**
India		11 989,71	**11 989,71**
Indonesia, Republic of	7 545,78		**7 545,78**
Italy	595,93		**595,93**
Jamaica	2 355,10		**2 355,10**
Japan		13 379,10	**13 379,10**
Kenya	1 935,02	6 518,19	**8 453,21**
Mexico	1 218,37	1 395,73	**2 614,10**
Morocco		1 038,77	**1 038,77**
New Zealand		1 309,84	**1 309,84**
Nicaragua	2 274,68		**2 274,68**
Papua New Guinea	1 206,94		**1 206,94**
Peru	1 478,34		**1 478,34**
Russia		1 989,38	**1 989,38**
Saudi Arabia	2 860,39		**2 860,39**
Sri Lanka		2 072,56	**2 072,56**
Tanzania, United Republic of	1 677,68		**1 677,68**
Turkey	686,84		**686,84**
United States	2 127,60		**2 127,60**
Yemen	826,56		**826,56**
Zimbabwe	1 331,42		**1 331,42**
Total	**82 550,63**	**94 178,70**	**176 729,33**

Figur 14.3 Visualisering laget med Power BI

Det vi skal se på i dette kapittelet er forskjellige typer visualiseringer, og hva de brukes til.

14.2 Tabeller

Resultatet av en SELECT-setning er en tabell, selv om de færreste tenker på det som noe slikt. Det som kjennetegner en tabell, er at den har kolonner og rader, og slik presenteres da også resultatet av SQL-spørringer:

dato	Solgt	Antall_ordre
2005-10-01	2302.26	25
2005-10-02	442.70	11
2005-10-03	589.18	7
2005-10-04	899.26	11
2005-10-05	1240.35	14
2005-10-06	1305.39	13
2005-10-07	828.76	15
2005-10-10	825.64	14
2005-10-11	1384.53	18
2005-10-12	1776.60	22
2005-10-14	2202.75	23
2005-10-15	479.67	5

Figur 14.4 Resultatet av en SELECT-setning i MySQL Workbench

Både rapporten laget med BIRT og presentasjonen laget med Power BI er enkle tabeller. Her er hver rad summen av salg for henholdsvis produksjonsland (Power BI) og dato (BIRT). Hver oppsummering ligger i en post i tabellen.

Dette er de enkleste formene for tabeller, og vi sier at de er endimensjonale. Dimensjonen er henholdsvis produsentland og dato.

I visualisering er det kanskje vanligere å bruke to eller flere dimensjoner i en tabell. Her er vist en todimensjonal *krysstabell*, laget med Excel:

Summer av quantity	Kolonneetiketter		
Radetiketter	C	T	Totalsum
April	383,2	713,6	1096,8
November	736,5	1363,9	2100,4
December	715,9	1464,2	2180,1
February	704,5	1085,1	1789,6
January	711	1277,1	1988,1
March	658,3	1075,5	1733,8
October	745,1	1087	1832,1
Totalsum	4654,5	8066,4	12720,9

Figur 14.5 Typisk krysstabell

I denne tabellen er kolonnene varegruppene pluss totalsum, mens radene er månedsnavn. Inne i tabellene er antall solgt summert opp.

Krysstabellene kan også bruke flere dimensjoner. Nedenfor er vist en hvor radene har varegrupper, men for hver varegruppe er det videre delt inn i enkeltvare. Kolonnene er årstall pluss total:

Category	Type	2001	2002	2003	Grand total
Fruit	Apples	150	153	162	**465**
	Bananas	332	336	344	**1012**
	Pears	267	266	279	**812**
	Subtotal	**749**	**755**	**785**	**2289**
Vegetables	Cucumber	140	141	152	**433**
	Lettuce	246	245	258	**749**
	Tomatoes	156	161	168	**485**
	Subtotal	**542**	**547**	**578**	**1667**
Grand total		**1291**	**1302**	**1363**	**3956**

Figur 14.6 En litt mer avansert krysstabell (docs.tibco.dom)

Krysstabeller er en klassisk måte å presentere data på innen statistikk, økonomi, forskning m.m.

I neste omgang kommer vi til pivottabeller. Dette er også krysstabeller, men nå laget med et dataverktøy der brukeren kan bytte om på x- og y-aksen, velge andre data for summeringer, legge til eller fjerne dimensjoner m.m. Med Excel kan vi også definere hierarkier i dimensjonene, noe som gir muligheten for drilling. Dette skal vi se nærmere på i neste kapittel.

	Kolonneetiketter					
	⊞Kaffe		⊞Te		Totalt Sum av actual_sold	Totalt Sum av quantity
Radetiketter	Sum av actual_sold	Sum av quantity	Sum av actual_sold	Sum av quantity		
⊟2005	20402,06	2788	35925,54	3558	56327,6	6346
⊞November	6848,16	916	12253,66	1124	19101,82	2040
⊞December	6575,34	942	13984,41	1377	20559,75	2319
⊞October	6978,56	930	9687,47	1057	16666,03	1987
⊟2006	22476,4	3105	37756,19	3718	60232,59	6823
⊞April	3264,95	481	6042,49	641	9307,44	1122
⊟February	6331,76	897	9566,87	884	15898,63	1781
01.02.2006	76	12			76	12
03.02.2006	147,8	23	169,68	23	317,48	46
04.02.2006	118,75	22	488,15	45	606,9	67

Figur 14.7 Pivottabell med drilling i Excel

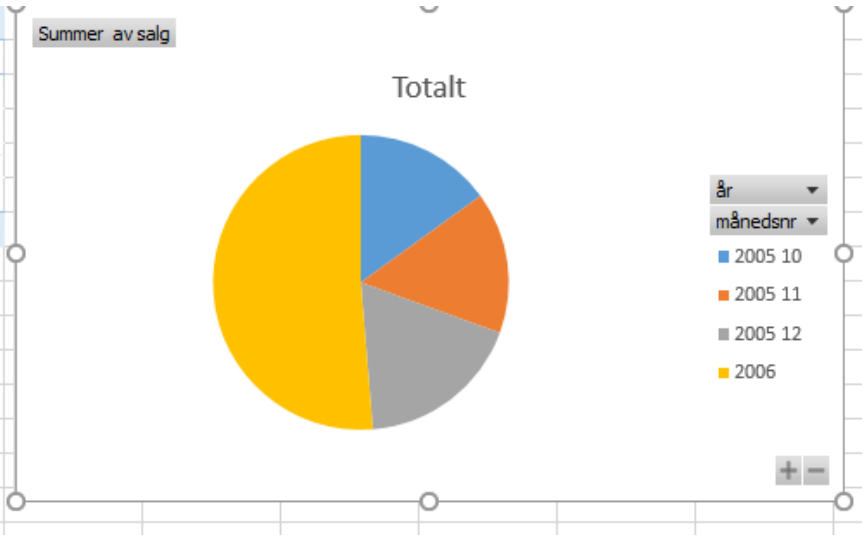
14.3 Grafiske presentasjoner

Selv om tabeller forenkler tolkingen av data i stor grad, er det ofte vi også bruker grafiske presentasjoner. Disse kan enten erstatte tabellene, eller brukes sammen med dem. Det finnes en rekke forskjellige diagramtyper, hver til sitt bruk. Vi skal her se på noen av de mest relevante for presentasjon av forretningsdata. Både Excel og Power BI har rike muligheter for å lage grafikk.

14.3.1 Sektordiagrammer

Et sektordiagram (Pie Chart) viser fordelingen av verdier på en måte som er velkjent fra dagliglivet: som en kake eller pizza som er stykket opp (populært kalles diagramtypen ofte kakediagram). Størrelsen på stykket viser summen av en eller annen variabel. Nedenfor er vist en pivottabell for coffeemerchant, der vi for år 2005 har drillet ned på salg per måned, mens 2006 viser hele årets salg:

Radetiketter	Summer av salg
2005	86085,47
10	26521,45
11	27468,05
12	32095,97
2006	90643,86
Totalsum	176729,33



Figur 14.8 Pivottabell med kakediagram

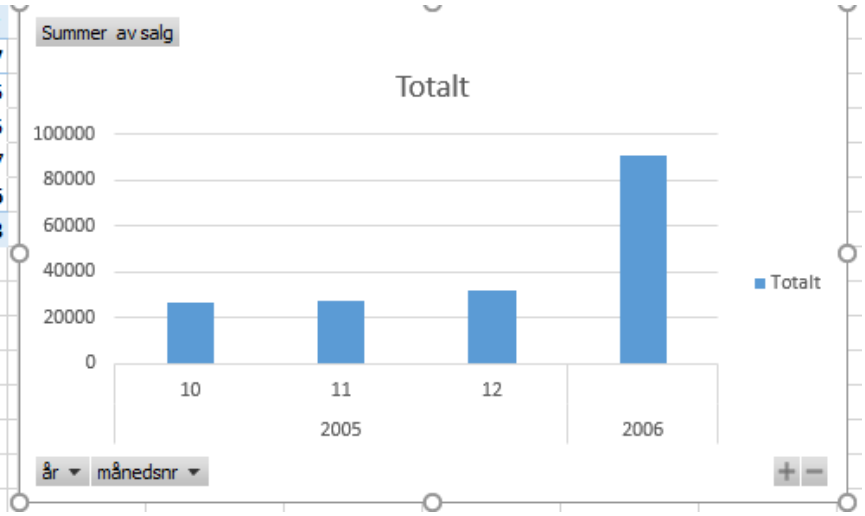
Kakediagrammer har en lang historie. Mest kjent for tidlig bruk av kakediagrammer var Florence Nightinggale da hun skulle overbevise politikere om mønstere i dødelighet av soldatene i krigen mellom Storbritannia og Tyrkia på midten av 1800-tallet:

Kakediagrammer er mye brukt, og de finnes også i flere varianter.

14.3.2 Søylediagrammer

Et søylediagram har to akser: en som viser antall og en som viser kategorier. Kategoriene kan være alt vi ønsker målinger for. Nedenfor er vist et søylediagram der målingene er for måneder i 2005 og alle måneder for 2006, altså nøyaktig det samme som for kakediagrammet i figur 14.8. Her angir y-aksen verdiene: (sortering!)

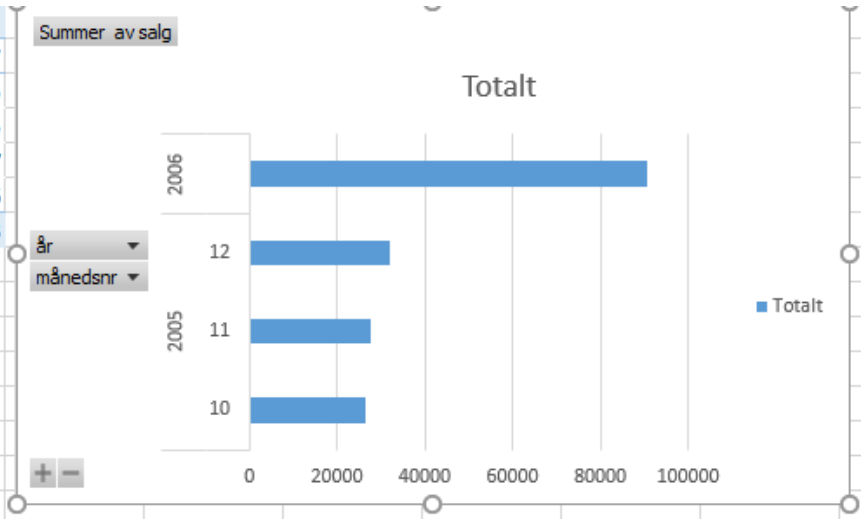
Radetiketter	Summer av salg
2005	86085,47
10	26521,45
11	27468,05
12	32095,97
2006	90643,86
Totalsum	176729,33



Figur 14.11 Samme data som i figur 14.8 vist som stolpediagram

Vi kan også ha liggende søyler, dvs at verdiene ligger langs x.aksen:

Radetiketter	Summer av salg
2005	86085,47
10	26521,45
11	27468,05
12	32095,97
2006	90643,86
Totalsum	176729,33

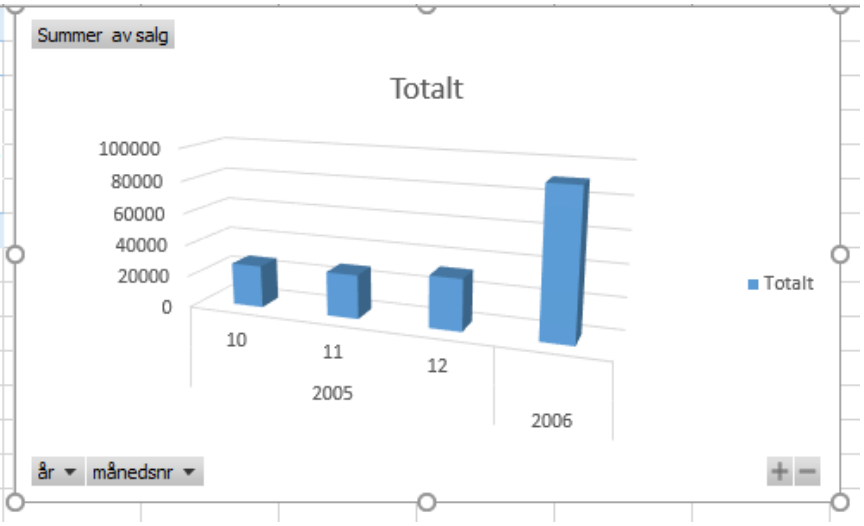


Figur 14.12 Liggende søyler

I norsk utgave av Excel kalles disse diagrammene stolpediagrammer. Strengt tatt er stolpediagrammer diagrammer uten bredde på søylene, dvs bare en strek.

Verktøy som Excel tilbyr også muligheten for mer fancy diagrammer. Her er et søylediagram i 3D-utgave:

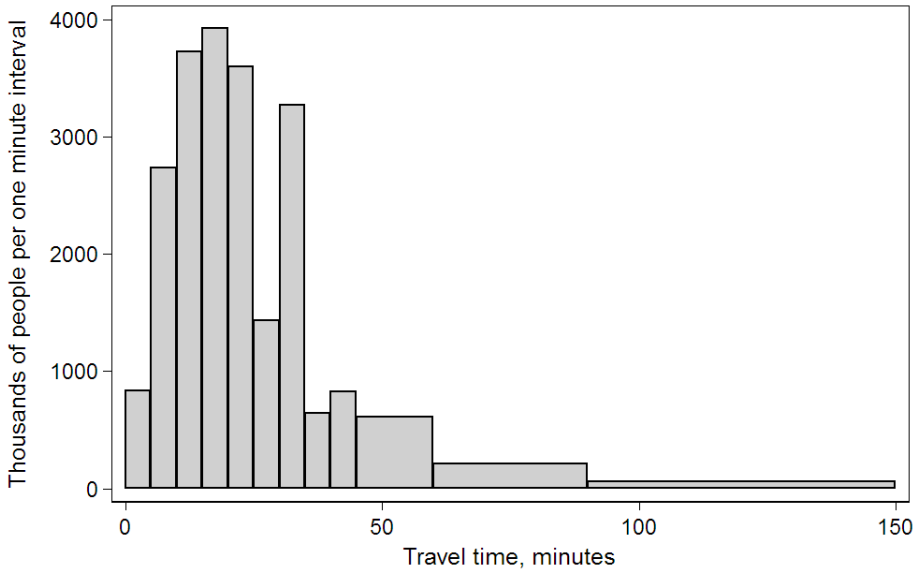
Radetiketter	Summer av salg
2005	86085,47
10	26521,45
11	27468,05
12	32095,97
2006	90643,86
Totalsum	176729,33



Figur 14.13 3D søylediagram

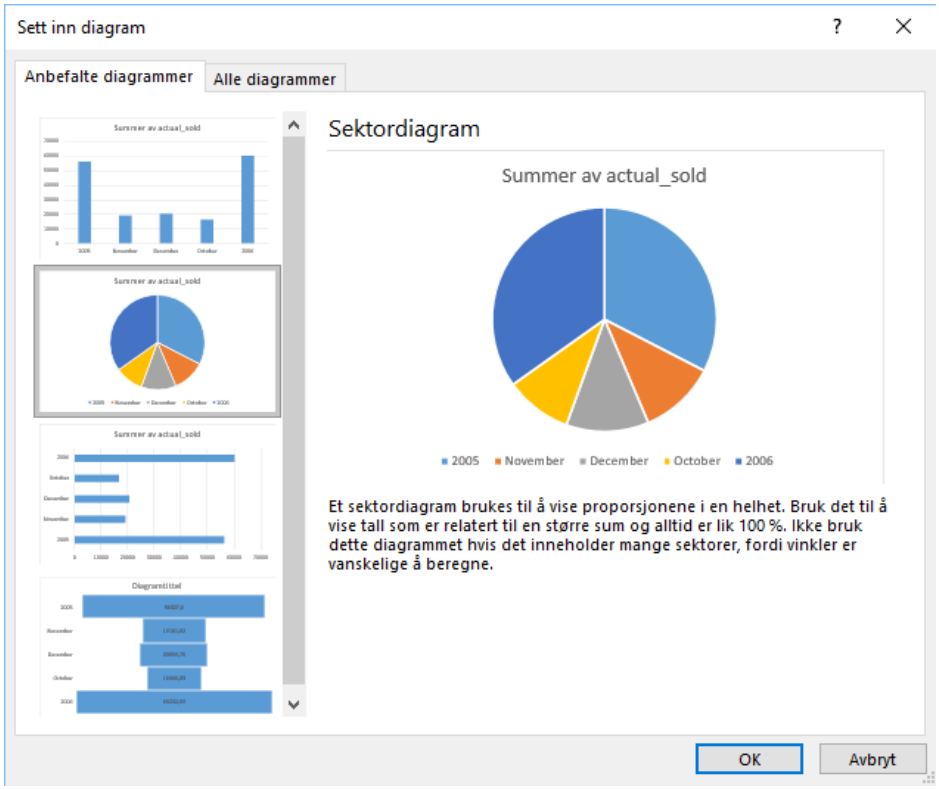
Matematikere og statistikere er ikke så begeistret for denne måten å visualisere på, fordi det gjør det vanskeligere å matche søylene mot verdiene på y-aksen.

Søylediagrammer kalles noen ganger feilaktig for *histogrammer*. Et histogram ligner, men i disse kan bredden på de enkelte søyler variere innen diagrammet. Bredden på søylene bestemmes av klassebredden, som er forskjellen mellom minste og største verdi i et intervall. Oftest vil intervallene være like store, og søylene har da samme bredde. Men intervallene kan også være forskjellige, slik at søylene får ulik bredde. Det er best å overlate denne diagramtypen til statistikere.



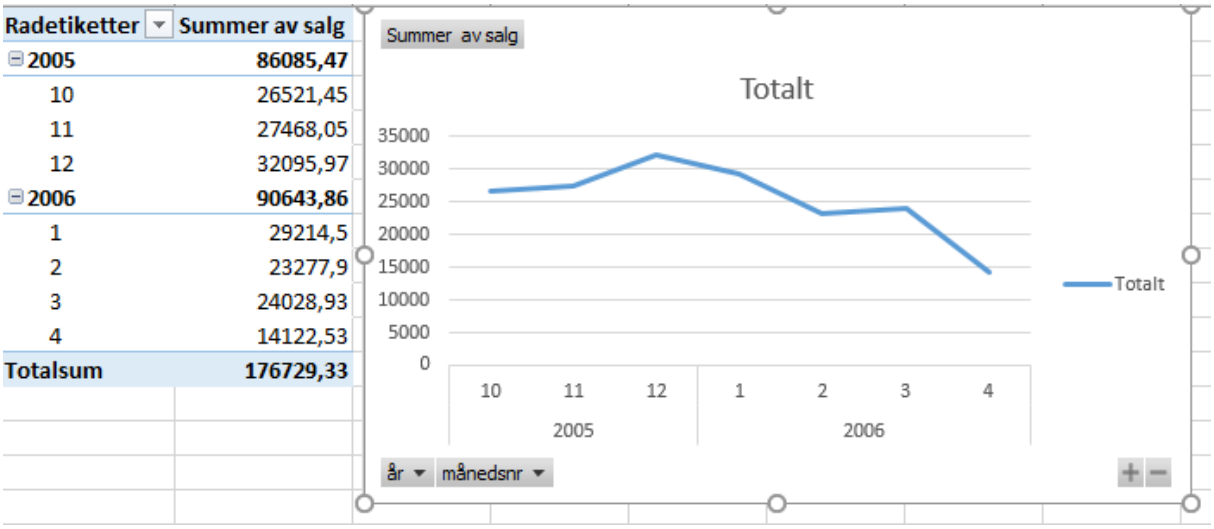
Figur 14.14 Histogram med forskjellige søylebredder (en.wikipedia.org/wiki/)

Med Excel kan vi velge å få anbefalt diagramtype. Vi får da også forklart hva diagramtypen er godt for. Nedenfor er vist anbefalingene for den datatabellen vi har brukt:



Figur 14.14 Anbefalinger til diagramtype i Excel

En annen diagramtype er linjediagram. Her er et eksempel, basert på salg i alle månedene i eksempeldatabasen Coffeemerchant:



Figur 14.15 Linjediagram. Her er alle månedene detaljert

Kapittel 15 Online Analytical Processing – OLAP

Online Analytical Processing – oftest omtalt som OLAP – ble i sin tid definert av Edgar F. Codd (han med relasjonsdatabasemodellen). Enkelt sagt kan OLAP defineres som ”The dynamic synthesis, analysis, and consolidation of large volumes of multi-dimensional data” (Connolly 2002). Begrepet OLAP dekker flere typer analyser:

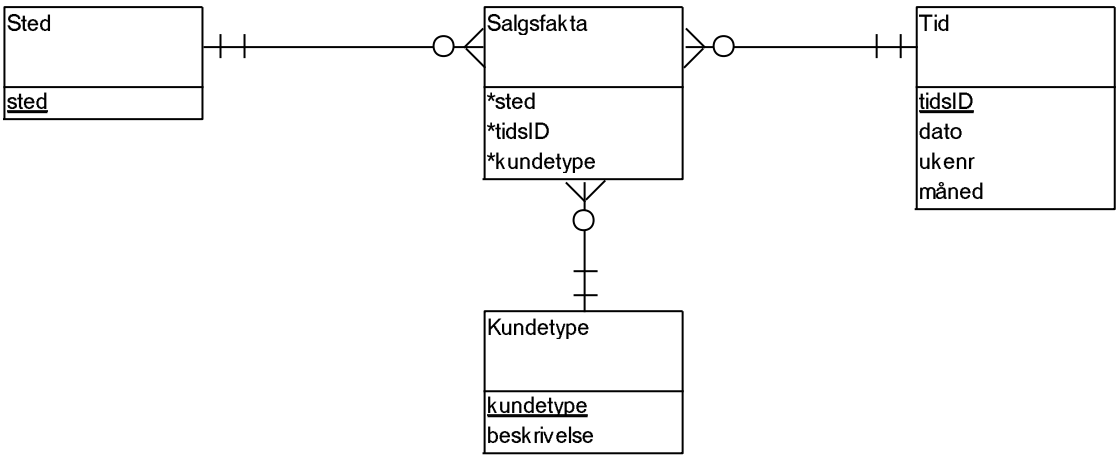
- Navigering og browsing (kalt “slicing and dicing”)
- Pivoting
- Drilling
- Utregninger
- Kompliserte modeller
- Tidsrekkeanalyse

Typisk er at analyse skjer langs flere dimensjoner, som produkt, tid, kunde osv., og det er vanlig å snakke om *multidimensjonale* data (Hobbs 2003). Codds beskrivelse av OLAP inkluderer også en multidimensjonal modell, i motsetning til bruk av den relasjonelle modellen (Dodge 1998). Med en multidimensjonal modell kan man forandre på rapporter ved å bytte om på dimensjoner. Ved å bytte om på dimensjoner, vil man få nye aggregeringer av data. Man skal også kunne selektere på enkeltdimensjoner. En enkel illustrasjon på denne måten å arbeide på er ved bruk av pivottabeller.

15.1 Multidimensjonale modeller

Vi kjenner allerede multidimensjonale data fra stjerneskjemaene, og vi har implementert dem relasjonelt med SQL. Med OLAP vil man imidlertid ofte ha en annen representasjon av data, enten i internminnet eller på disk. Mange OLAP-verktøy henter data fra en relasjonsdatabase og håndterer dem internt i egne datastrukturer. Dette er de såkalte ROLAP-verktøy (Relasjonell OLAP). Andre lagrer også data i sitt eget lagringsformat, som altså ikke er relasjonelt. Disse verktøyene kalles MOLAP (Multidimensjoal OLAP). Mer om dette lenger ned.

Vi skal se på hvordan vi kan fremstille et stjerneskjema relasjonelt og multidimensjonalt. La oss ta utgangspunkt i følgende stjerneskjema:



Figur 15.1 Dimensjonsmodell

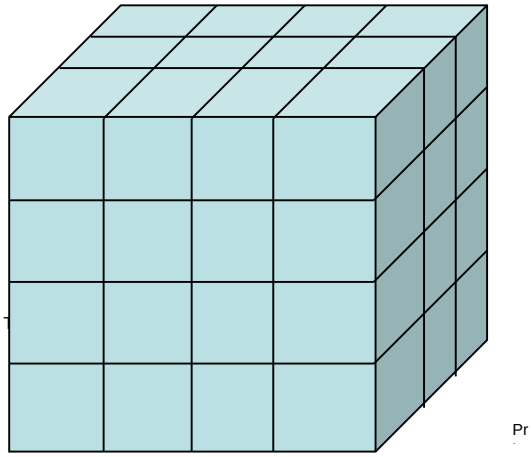
Ved en implementering etter relasjonsmodellen vil entitetstypen Salgsfakta bli en tabell på denne måten (i stedet for fremmednøkler er brukt fulle attributtnavn):

Sted	Måned	Kundetype	Omsetning
Oslo	Januar	Privatkunde	5.870
Oslo	Januar	Bedrifter	7.850
Oslo	Januar	Offentlig	2.654
Oslo	Februar	Privatkunder	4.960
Oslo	Februar	Bedrifter	8.569
Oslo	Mars	Offentlig	3.258

Figur 15.2 Dimensjonsmodell implementert som tabell

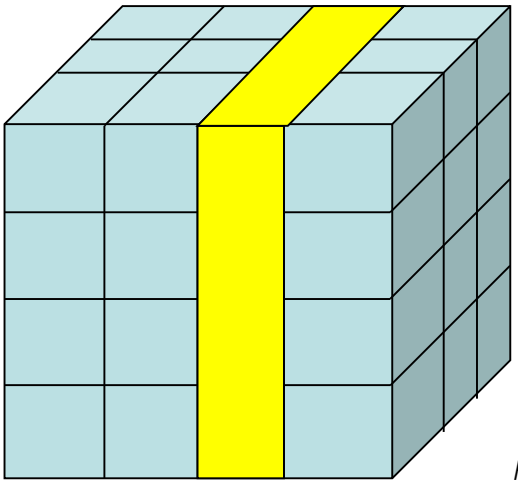
En **multidimensjonal** implementering som denne kan vi illustrere som en kube (det er tre dimensjoner, og en kube er tredimensjonal). Dette kan vi også illustrere med slageren "Rubiks kube" fra 1980-tallet:

Nedenfor er en multidimensjonal fremstilling av stjerneskjemaet vårt. Radene markert med gult i tabellen over finner vi igjen i kuben.



Figur 15.3 Multidimensjonale data fremstilt som kube

Typisk bruk av kuber er ”slicing” og ”dicing”. ”Slicing” kan illustreres som at vi ”skjærer ut skiver” av kuben (slice betyr skive). I figur 15.26 er dette illustrert. I dette tilfellet har vi ”skåret ut” måneden mars fra kuben. Den skiven vi da får er en todimensjonal tabell med kundetype og sted.



Figur 15.4 Illustrasjon av slicing

Vi kan illustrere resultatet som at vi har skåret ut en skive for måneden mars. Skiven ser slik ut:

	Oslo	Bergen	Trondheim	Skien
Offentlige virksomheter	7.547	95	958	58
Bedrifter	10.520	125	1.950	214
Privatkunder	5.560	65	1.810	89

Figur 15.5 "Utskårne" data

"Dicing vil si at vi deretter kan se den utskårne skiven fra forskjellige sider. I stedet for å se skiven slik det er vist i figur 15.6, kan vi i stedet se den slik:

	Offentlige virksomhet	Bedrift	Privatkunder
Oslo	**7.547**	**10.520**	**5.560**
Bergen	95	125	65
Trondheim	958	1.950	1.810
Skien	58	214	89

Figur 15.6 Dicing av skiven fra figur 15.6

Resultatet av "slicing and dicing" er altså en krysstabell. Når vi kan dreie aksene til en slik tabell, slik det er vist foran, kaller vi det ***pivoting***. I utgangspunktet er tabellen todimensjonal, men ved å legge inn **drilling** langs hierarkier, i tabellen, blir den flerdimensjonal.

15.2 ROLAP, MOLAP og HOLAP

Som nevnt over er det forskjellige måter å implementere OLAP på. De første OLAP-verktøyene benyttet en ikke-relasjonell database for lagring av data. Slike databaser knytter målinger til dimensjonsdata ved bruk av multidimensjonale arrayer. Det er dette som utgjør kuben. Et eksempel på et slikt OLAP-verktøy er IBM Cognos TM1.

Multidimensjonale OLAP-verktøy har mange fordeler:

- De kan være svært raske ved prosessering
- Svært kompakte databaser når antallet dimensjoner er lavt

Men de har også noen ulemper:

- Lasting av data kan ta svært lang tid med store datamengder
- Et fenomen kjent som «dataeksplosjon» kan skje med store datamengder og mange dimensjoner: databasen kan plutselig doble seg i størrelse

Relasjonell OLAP bygger på at data ligger i en relasjonsdatabase. ROLAP-verktøyet bygger typisk en datakube i internminnet, basert på spesifikasjoner som er gjort. Typisk er kuben representert ved XML-script. Mondrian er et slikt verktøy.

Fordeler med ROLAP:

- Relasjonsdatabaser er best på å håndtere store datamengder
- Kortere lastetid enn med MOLAP
- Databasen kan analyseres med SQL. De store databasesystemene (Oracle, SQL Server, DB2) har egne tillegg til SQL for OLAP, som CUBE og ROLLUP

Selvsagt er det også ulemper:

- Langsommere en MOLAP
- MOLAP-verktøy kan nyttiggjøre en del spesialiteter for analyse som ikke er mulig med ROLAP

Endelig har vi verktøy som kombinerer egenskapene til MOLAP og ROLAP. Disse kalles HOLAP, for Hybrid OLAP. Disse bruker en relasjonsdatabase til detaljdata og en multidimensjonal database til aggregerte data. Oracle Enterprise Server har en slik løsning.

15.3 Bruk av pivottabeller i Excel

Data fra en OLAP-analyse fremstilles gjerne i krysstabeller. Drilling foretas direkte i krysstabellen. En spesiell type krysstabeller er pivottabeller.

Pivot betyr ”å dreie” eller ”å dreie om”, og pivottabeller er nettopp tabeller der man enkelt kan ”dreie om” på dimensjoner. Det finnes mange verktøy for å lage pivottabeller,

blant annen har Oracle sitt eget verktøy for dette (Discoverer). Vi skal her se på hvordan det gjøres med et annet velkjent verktøy, nemlig Excel regneark.

15.3.1 Importere data

Vi skal i første omgang bruke en faktatabell vi har laget på¨forhånd i MySQL, og som inneholder flere av dimensjonenes attributtverdier. Blant disse har vi hierarkiene varetype – varenavn og år – måned – dato. Strukturen til denne tabellen ser slik ut:

Column Name	Datatype	NOT NULL	AUTO INC	Flags	Default Value
fact_pk	INTEGER			UNSIGNED ZEROFILL	NULL
time_sk	INTEGER			UNSIGNED ZEROFILL	NULL
dato	DATE				NULL
maanedsnavn	CHAR(9)			BINARY ASCII UNIC	NULL
aar	INT(4)			UNSIGNED ZEROFILL	NULL
inventory_sk	INTEGER			UNSIGNED ZEROFILL	NULL
name	VARCHAR(40)			BINARY	NULL
itemType	VARCHAR(1)			BINARY	NULL
customer_sk	INTEGER			UNSIGNED ZEROFILL	NULL
customername	VARCHAR(60)			BINARY	NULL
employee_sk	INTEGER			UNSIGNED ZEROFILL	NULL
employeename	VARCHAR(60)			BINARY	NULL
orderno	INTEGER			UNSIGNED ZEROFILL	NULL
quantity	INTEGER			UNSIGNED ZEROFILL	NULL
price	FLOAT(6,2)			UNSIGNED ZEROFILL	NULL
actual_price	FLOAT(6,2)			UNSIGNED ZEROFILL	NULL
actual_sold	FLOAT(10,2)			UNSIGNED ZEROFILL	NULL

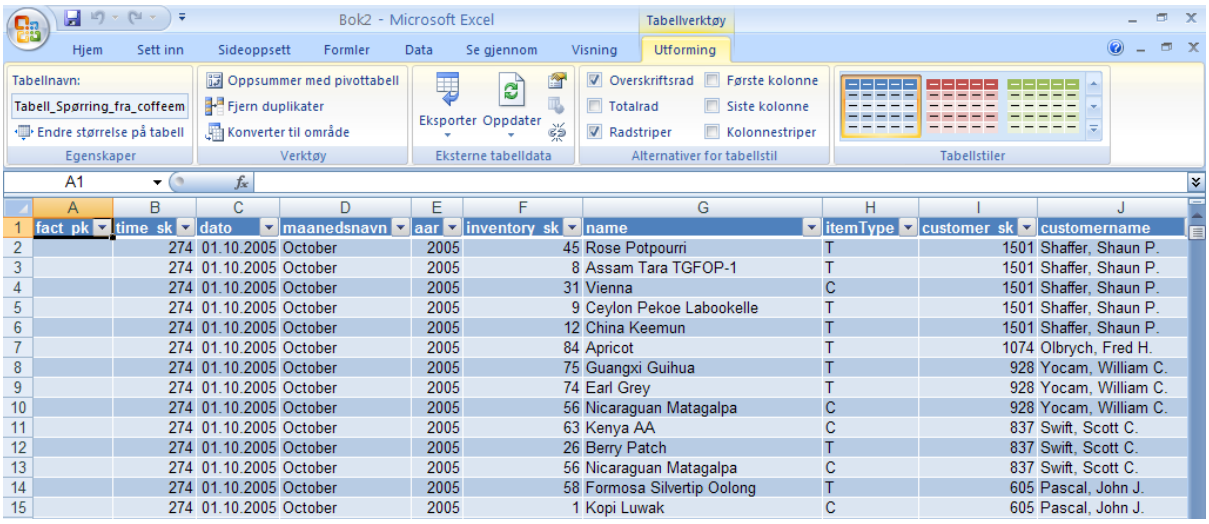
Figur 15.7 Strukturen til tabellen som skal importeres

Vi skal senere lage en ODBC-kobling til databasen coffeemerchant_dw. Denne skal vi bruke til å koble Excel direkte til databasen.

Strukturen til tabellen som er grunnlaget for Excel-filen er slik:

```
USE coffeemerchant_dw;
CREATE TABLE fact_sales_dimensions (
  fact_pk INT,
  time_sk INT,
  dato DATE,
  inventory_sk INT,
  name VARCHAR(40),
  customer_sk INT,
  customername VARCHAR(60),
  employee_sk INT,
  employeename VARCHAR(60),
  orderno INT,
  quantity INT,
  price FLOAT(6,2),
  actual_price FLOAT(6,2),
  actual_sold FLOAT(10,2));
```

Vi skal senere se på hvordan vi lager en Excel-fil av denne tabellen. Foreløpig skal vi bare bruke filen coffeemerchant_dw. Regnearket ser slik ut:



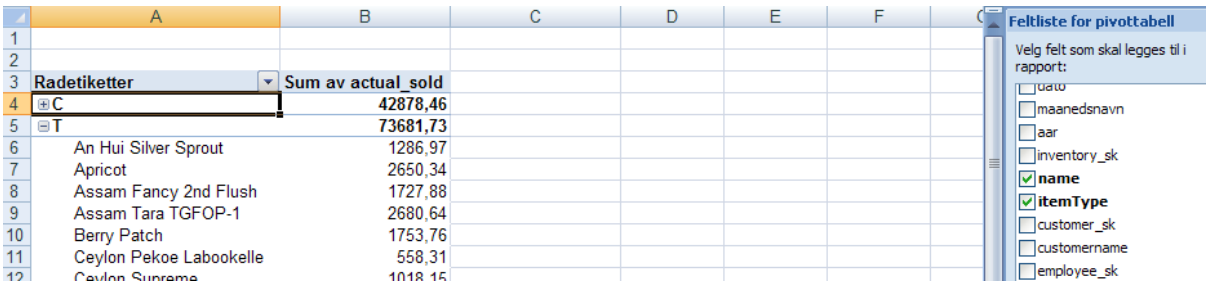
	A	B	C	D	E	F	G	H	I	J
1	fact_pk	time_sk	dato	maanedsnavn	aar	inventory_sk	name	itemType	customer_sk	customername
2		274	01.10.2005	October	2005	45	Rose Potpourri	T	1501	Shaffer, Shaun P.
3		274	01.10.2005	October	2005	8	Assam Tara TGFOP-1	T	1501	Shaffer, Shaun P.
4		274	01.10.2005	October	2005	31	Vienna	C	1501	Shaffer, Shaun P.
5		274	01.10.2005	October	2005	9	Ceylon Pekoe Labookelle	T	1501	Shaffer, Shaun P.
6		274	01.10.2005	October	2005	12	China Keemun	T	1501	Shaffer, Shaun P.
7		274	01.10.2005	October	2005	84	Apricot	T	1074	Olbrych, Fred H.
8		274	01.10.2005	October	2005	75	Guangxi Guihua	T	928	Yocam, William C.
9		274	01.10.2005	October	2005	74	Earl Grey	T	928	Yocam, William C.
10		274	01.10.2005	October	2005	56	Nicaraguan Matagalpa	C	928	Yocam, William C.
11		274	01.10.2005	October	2005	63	Kenya AA	C	837	Swift, Scott C.
12		274	01.10.2005	October	2005	26	Berry Patch	T	837	Swift, Scott C.
13		274	01.10.2005	October	2005	56	Nicaraguan Matagalpa	C	837	Swift, Scott C.
14		274	01.10.2005	October	2005	58	Formosa Silvertip Oolong	T	605	Pascal, John J.
15		274	01.10.2005	October	2005	1	Kopi Luwak	C	605	Pascal, John J.

Figur 15.8 Regnearket coffeemerchant_dw

15.3.2 Lage pivottabeller

Vi skal nå lage en Pivottabell. Gå til arkfanen Sett inn, og velg Pivottabell. Legg denne i et nytt ark.

Vi skal nå lage en drill-down pivottabell ved å velge itemtype og name (som er varenavn), og så actual_sold (som er totalt salg for en varelinje etter at rabatt er trukket fra). Resultatet er en tabell der det øverste nivået ser slik ut:



	A	B
3	Radetiketter	Sum av actual_sold
4	⊞C	42878,46
5	⊟T	73681,73
6	An Hui Silver Sprout	1286,97
7	Apricot	2650,34
8	Assam Fancy 2nd Flush	1727,88
9	Assam Tara TGFOP-1	2680,64
10	Berry Patch	1753,76
11	Ceylon Pekoe Labookelle	558,31

Figur 15.9 Pivottabellen vi skal lage

Vi kan nå klikke på + foran C, og får da en drilling ned til varenavn:

Radetiketter	Sum av actual_sold
C	42878,46
Brazil Bourbon Santos	1644,67
Brazil Sul De Minas Cerra	1167,72
Celebes Kalossi	2191,28
Chanchamayo	1478,34
Columbia Bucaramanga Especial	1461,87
Columbia Supremo	1079,96
Costa Rica La Manita	1700
Costa Rica Tarrazu	970,14
Ethiopia Harrar	1439,2
Ethiopia Mokas	895,4

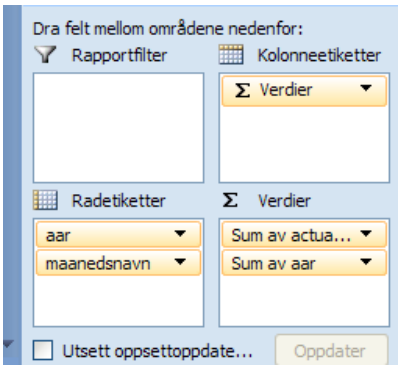
Figur 15.10 Drilling i pivottabellen

Vi kan prøve oss med et nytt hierarki. Her er en drilling-tabell med år og måned:

	Verdier	
Radetiketter	Sum av actual_sold	Sum av aar
2005	56327,6	1225055
November	19101,82	403005
December	20559,75	437090
October	16666,03	384960
2006	60232,59	1352044
April	9307,44	236708
February	15898,63	367098
January	19248,86	415242
March	15777,66	332996
Totalt	116560,19	2577099

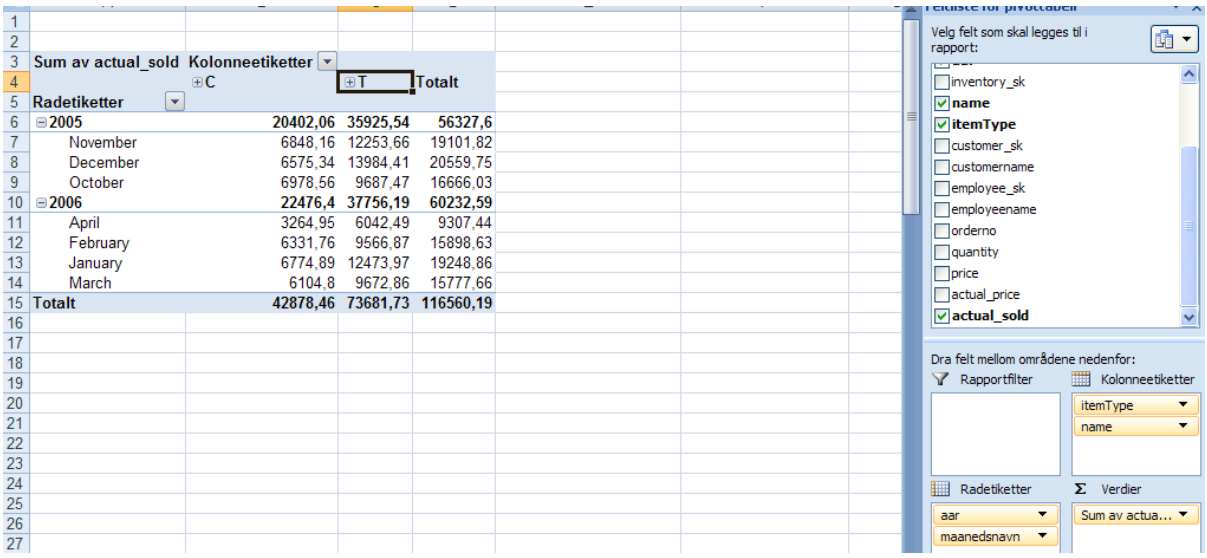
Figur 15.11 En annen pivottabell laget med de samme dataene

Merk at vi kan kontrollere hierarkiene i radvinduet nederst til høyre. Noen ganger klarer ikke Excel dette selv, og da må vi dra et attributt ned i radvinduet og legge det inn på riktig plass. Vi kan også bytte om på rekkefølgen i dette vinduet, dersom hierarkiet er blitt feil:



Figur 15.12 Vindu for valg av rader og kolonner

Excel legger automatisk attributtene i rader. Ofte ønsker vi i stedet en krysstabell, med attributtverdier også i kolonnene. Vi drar da attributtene på plass i kolonnevinduet, og det som skal summeres i summeringsvinduet. Her har vi laget en krysstabell med år – måned i radene og varetype – varenavn i kolonnene:



Figur 15.13 Pivottabell med definisjonsvinduet til høyre

Vi kan nå gå opp til årsnivå på radene:

Sum av actual_sold	Kolonneetiketter		
	C	T	Totalt
Radetiketter			
2005	20402,06	35925,54	56327,6
2006	22476,4	37756,19	60232,59
Totalt	42878,46	73681,73	116560,19

Figur 15.14 Rollup til årsnivå

Eller vi kan drille helt ned både på rader og kolonner:

Sum av actual_sold	Kolonneetiketter			
	C			
Radetiketter	Brazil Bourbon Santos	Brazil Sul De Minas Cerra	Celebes Kalossi	Chanchamayo
2005	624,99	937,95	1163,76	823,34
November	277,8	347,06	332,8	191,26
December	148,4	471,01		275,76
October	198,79	119,88	830,96	356,32
2006	1019,68	229,77	1027,52	655
April	219,05	66,6	292,76	
February	365,15	133,57	10,4	78,6
January	86,4	29,6	443,04	203,05
March	349,08		281,32	373,35
Totalt	1644,67	1167,72	2191,28	1478,34

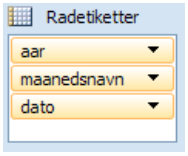
Figur 15.15 Drilling ned til enkeltvarer

Endelig kan vi skrive om kategorinavnene C og T til det mer forståelige Kaffe og Te:

Sum av actual_sold	Kolonneetiketter		
	Kaffe	Te	Totalt
Radetiketter			
2005	20402,06	35925,54	56327,6
November	6848,16	12253,66	19101,82
December	6575,34	13984,41	20559,75
October	6978,56	9687,47	16666,03
2006	22476,4	37756,19	60232,59
April	3264,95	6042,49	9307,44
February	6331,76	9566,87	15898,63
January	6774,89	12473,97	19248,86
March	6104,8	9672,86	15777,66
Totalt	42878,46	73681,73	116560,19

Figur 15.16 Pivottabell med våre egne kolonneoverskrifter

Vi kan nå legge til et nivå til i tidsdimensjonen. Vi drar dato ned i radvinduet:



Figur 15.17 Et nytt nivå er lagt til

Resultatet er nå dette:

Sum av actual_sold	Kolonneetiketter		
	Kaffe	Te	Totalt
Radetiketter			
2005	20402,06	35925,54	56327,6
November	6848,16	12253,66	19101,82
01.11.2005	11,2	97,24	108,44
02.11.2005	30,3	98,1	128,4
04.11.2005	209,05	658,43	867,48
05.11.2005	16,5	275,5	292
07.11.2005	277,2	486,19	763,39
08.11.2005	291,46	483,9	775,36

Figur 15.18 Pivottabellen med et nytt nivå lagt inn

Vi kan selvfølgelig måle flere verdier samtidig. Her er antall solgt lagt til:

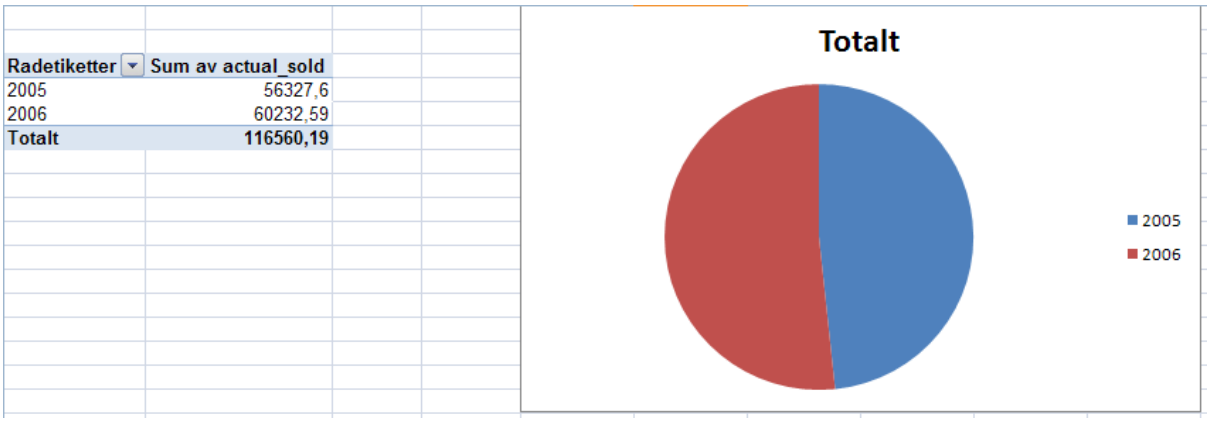
	Kolonneetiketter					
	Kaffe		Te		Totalt Sum av actual_sold	Totalt Sum av quantity
Radetiketter	Sum av actual_sold	Sum av quantity	Sum av actual_sold	Sum av quantity		
2005	20402,06	2788	35925,54	3558	56327,6	6346
November	6848,16	916	12253,66	1124	19101,82	2040
December	6575,34	942	13984,41	1377	20559,75	2319
October	6978,56	930	9687,47	1057	16666,03	1987
2006	22476,4	3105	37756,19	3718	60232,59	6823
April	3264,95	481	6042,49	641	9307,44	1122
February	6331,76	897	9566,87	884	15898,63	1781
01.02.2006	76	12			76	12
03.02.2006	147,8	23	169,68	23	317,48	46
04.02.2006	118,75	22	488,15	45	606,9	67

Figur 15.19 Pivottabellen med en ny måling (antall) lagt til

Vi har nå laget en krysstabell der vi måler både hvor mange som er solgt, og hvor mye det er solgt for, og der vi kan drille ned fra år til måned til dag og fra varetype til varenavn. Dette er et enkelt verktøy som kan gi mye verdifull informasjon.

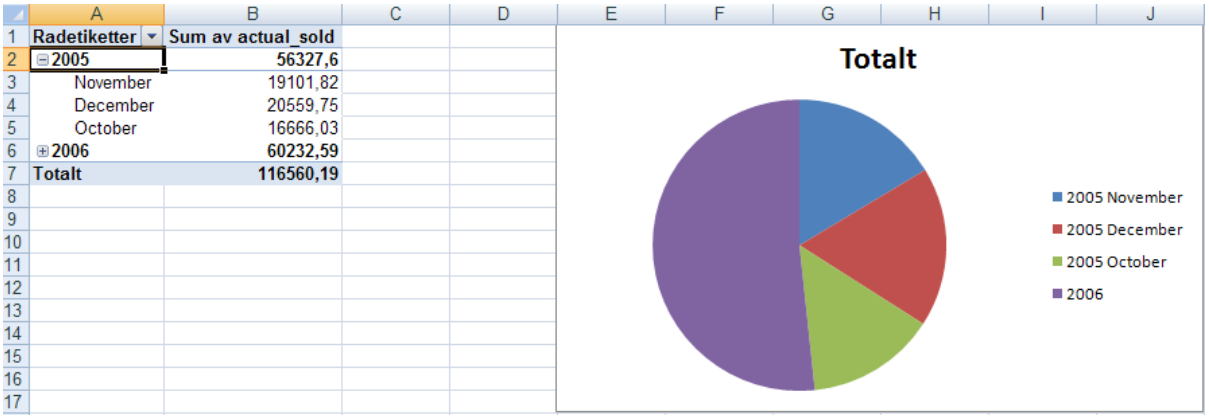
15.3.3 Diagrammer

Vi kan også lage diagrammer på en lignende måte. Vi velger Pivot-diagram, og legger år i aksevinduet og actual_sold i verdivinduet. Resultatet er dette:



Figur 15.20 Pivottabell med diagram

Også her kan vi legge inn drilling. Legger vi inn måned som neste nivå i hierarkiet, ser det slik ut når det er drillet på 2005, men ikke 2006:



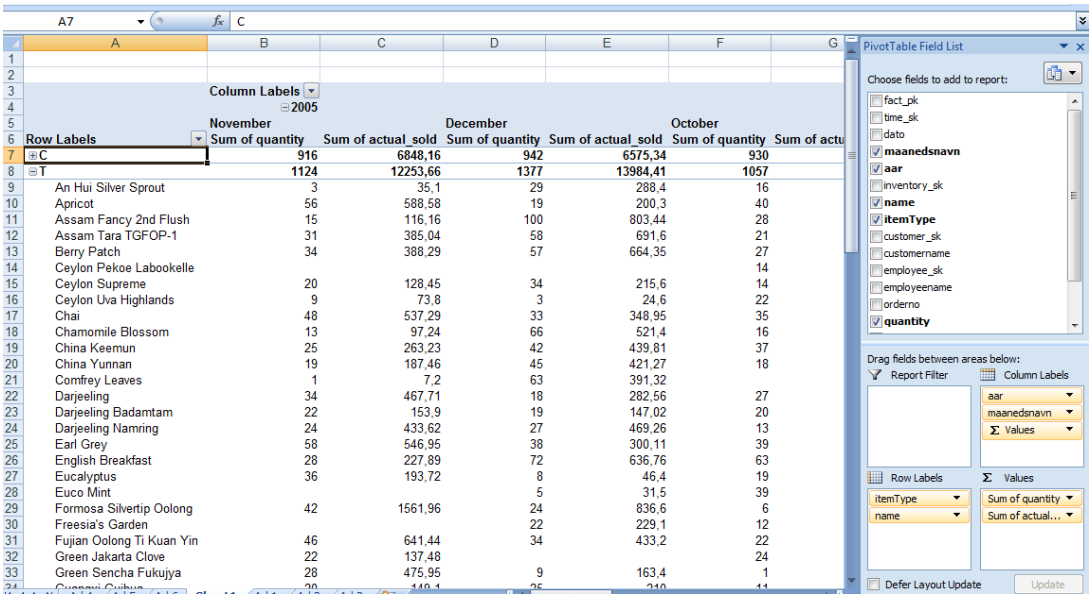
Figur 15.21 Drilling i tabellen gjenspeiles automatisk i diagrammet

Bruk av pivotdiagrammer brukes for å gjøre rapportene mer lesbare. Som vi ser kombineres de med selve tabellen.

Excel er imidlertid avhengig av at vi lager ferdig koblede tabeller som inneholder både fakta og de attributter vi vil bruke i drillingen. Dersom vi ønsker å arbeide direkte mot faktatabellene og dimensjonsmodellene i databasen, må vi bruke andre verktøy. Disse er basert på det som kalles multidimensjonale tabeller.

15.3.4 Utvalg langs dimensjonene

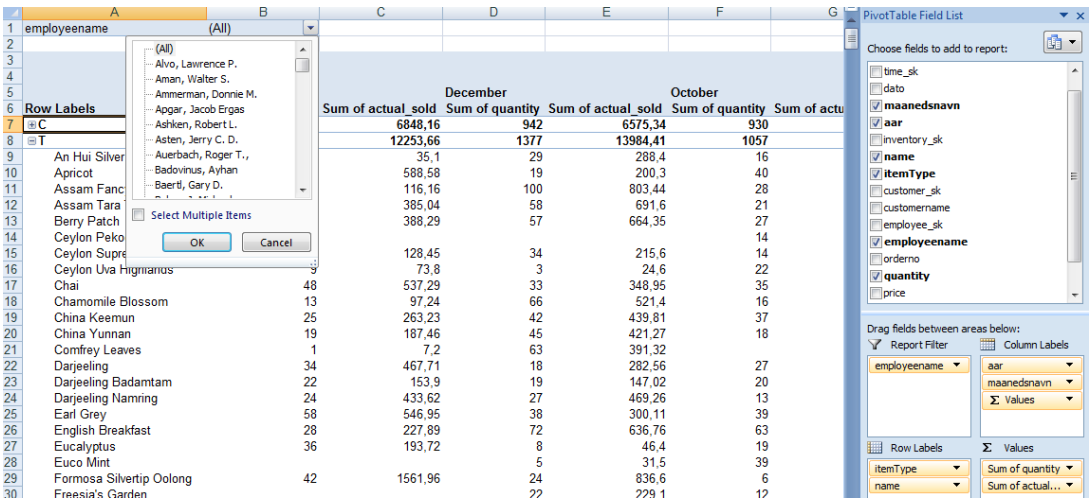
En viktig egenskap ved et pivot-verktøy er at vi kan gjøre et utvalg innenfor dimensjonene. La oss si at jeg ønsker å kontrollere hva den enkelte selger har solgt. Jeg begynner kanskje med en pivottabell som dette:



Row Labels	November Sum of quantity	November Sum of actual_sold	December Sum of quantity	December Sum of actual_sold	October Sum of quantity	October Sum of actu
C	916	6848,16	942	6575,34	930	
T	1124	12253,66	1377	13984,41	1057	
An Hui Silver Sprout	3	35,1	29	288,4	16	
Apricot	56	588,58	19	200,3	40	
Assam Fancy 2nd Flush	15	116,16	100	803,44	28	
Assam Tara TGFOP-1	31	385,04	58	691,6	21	
Berry Patch	34	388,29	57	664,35	27	
Ceylon Pekoe Labookelle					14	
Ceylon Supreme	20	128,45	34	215,6	14	
Ceylon Uva Highlands	9	73,8	3	24,6	22	
Chai	48	537,29	33	348,95	35	
Chamomile Blossom	13	97,24	66	521,4	16	
China Keemun	25	263,23	42	439,81	37	
China Yunnan	19	187,46	45	421,27	18	
Comfrey Leaves	1	7,2	63	391,32		
Darjeeling	34	467,71	18	282,56	27	
Darjeeling Badamtam	22	153,9	19	147,02	20	
Darjeeling Namring	24	433,62	27	469,26	13	
Earl Grey	58	546,95	38	300,11	39	
English Breakfast	28	227,89	72	636,76	63	
Eucalyptus	36	193,72	8	46,4	19	
Euco Mint			5	31,5	39	
Formosa Silvertip Oolong	42	1561,96	24	836,6	6	
Freesia's Garden			22	229,1	12	
Fujian Oolong Ti Kuan Yin	46	641,44	34	433,2	22	
Green Jakarta Clove	22	137,48			24	
Green Sencha Fukujya	28	475,95	9	163,4	1	

Figur 15.22 Pivottabell uten utvalg

Så legger jeg inn employeename i boksen kalt Import Filter (over radutvalg og til venstre for kolonneutvalg). Jeg kan da plukke ut en eller flere navngitte ansatte:

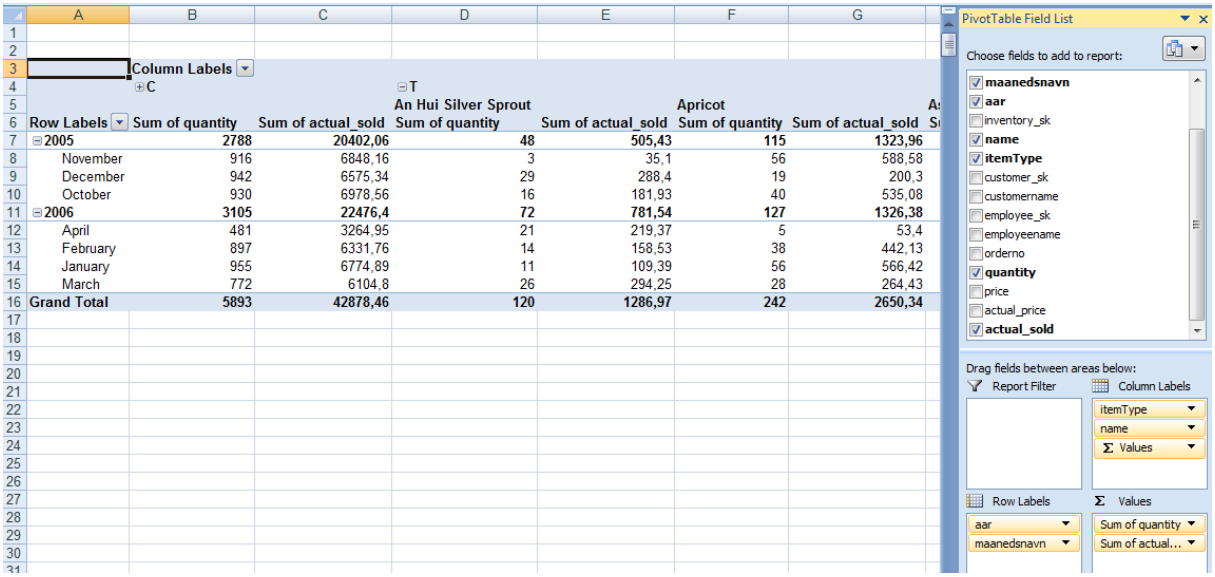


Figur 15.23 Valg av ansatt for rapportering

	A	B	C	D	E	F	G
1	employeename	Ammerman, Donnie M.					
2							
3		Column Labels					
4		2005		2005 Sum of quantity	2005 Sum of actual_sold	Total Sum of quantity	Total Sum of actual_sold
5		October					
6	Row Labels	Sum of quantity	Sum of actual_sold				
7	⊞C	22	178,73	22	178,73	22	178,73
8	⊟T	15	198,55	15	198,55	15	198,55
9	English Breakfast	5	68,5	5	68,5	5	68,5
10	Sixteen Herb Mu	10	130,05	10	130,05	10	130,05
11	Grand Total	37	377,28	37	377,28	37	377,28
12							

Figur 15.24 Resultatet av utvalg på en bestemt ansatt

Vi må ikke glemme dicing, dvs at vi kan bytte om på kolonner og rader. Her er utvalget fra figur 15.24 sett fra en annen vinkel:



Figur 15.25 Dicing eller pivoting av tabellen i figur 15.24

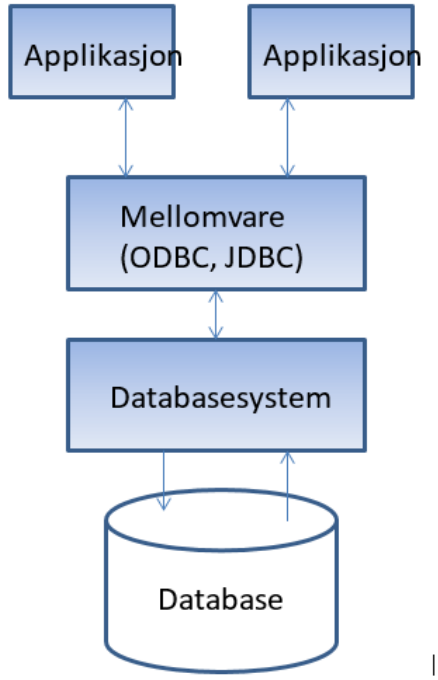
15.3.5 Lage spørringer med Microsoft Query

Microsoft Query er et tilleggsprogram som tillater brukere av Excel å definere sine egne spørringer på databasen. Dette betyr at vi ikke trenger ferdig sammenkoblede tabeller, men kan definere sammenkoblinger ved behov. Det går imidlertid adskillig langsommere enn om vi bruker ferdige tabeller.

Før vi kan bruke Excel til å spørre på databasen, må vi opprette en kobling mellom de to. Slike koblinger kjenner dere til fra før, i form av MySQL Connector NET for Visual Studio og JDBC for Java. Vi skal her bruke ODBC, som er en mer allment anvendbar kobling.

ODBC er et eksempel på mellomvare, dvs programvare som formidler kommunikasjon mellom to andre programmer. Hensikten er å skape uavhengighet mellom applikasjon og database. ODBC utgjør for applikasjonen et standard grensesnitt, definert av Microsoft. Fra databasens side er grensesnittet tilpasset det enkelte databasesystem. Dere kjenner et eksempel på mellomvare fra før: JDBC som brukes sammen med Java (og bare Java).

MySQL Connector NET er også mellomvare. ODBC er mer omfattende enn JDBC, og har ingen begrensninger når det gjelder språk. Vi kan illustrere dette slik:

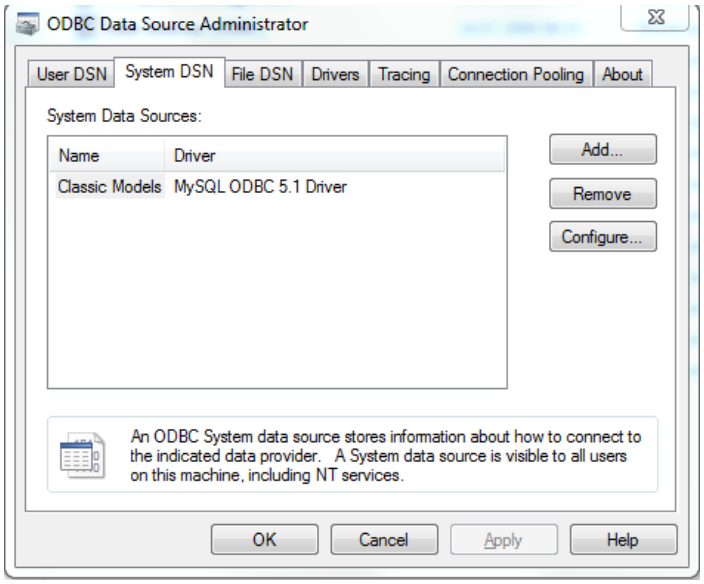


Vi laster ned MySQL Connector ODBC fra MySQL's nettsider:

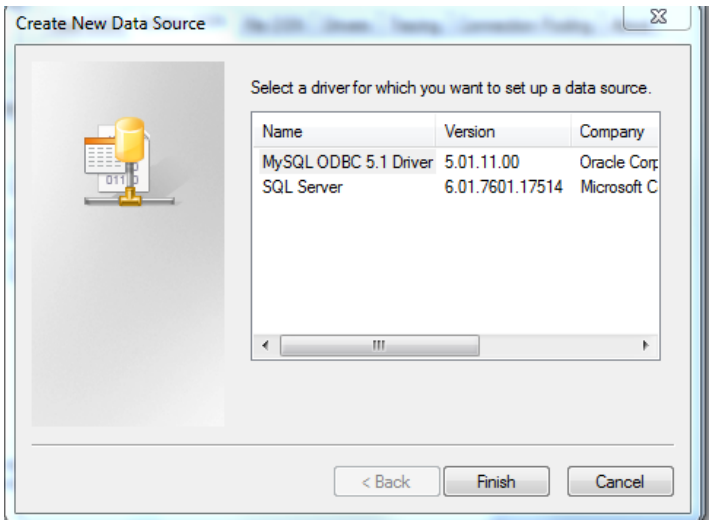
http://dev.mysql.com/downloads/connector/odbc/

Deretter kjører vi filen. Den blir lagt i en egen mappe under MySQL.

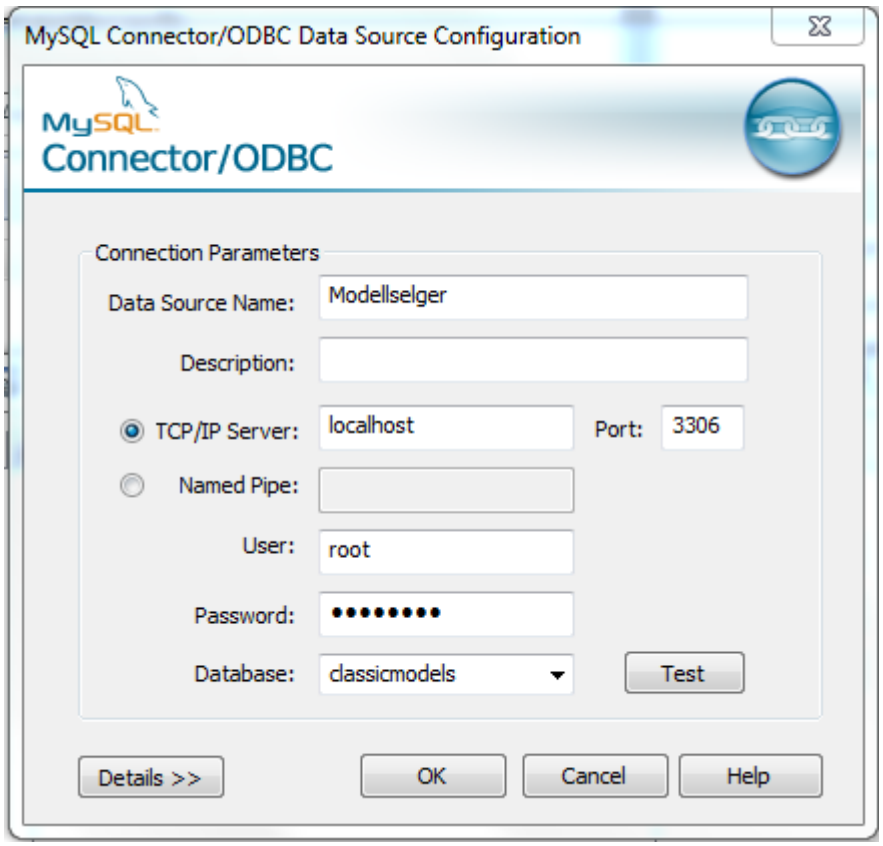
Vi går så inn i kontrollpanelet til Windows, velger Systems and Security (jeg har engelsk utgave) og så Administrative Tools. Der velger vi så Data Sources (ODBC). Slik ser det ut med en ferdig definert kobling (Datakilde i ODBC-terminologi). Koblinger definert som System DSN er tilgjengelige for alle brukere av maskinen.



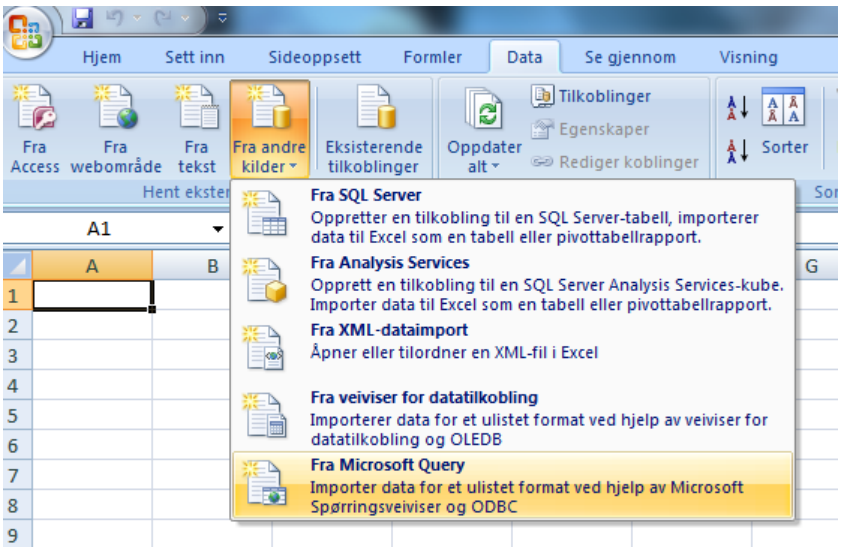
Vi kan legge til en ny kilde ved å trykke Add. Vi får da se tilgjengelige drivere:



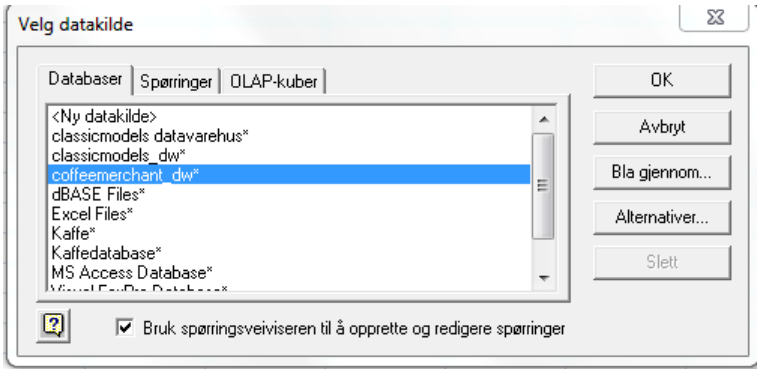
Vi velger MySQL-driveren, og kan så definere forbindelsen:



Nå er koblingen ferdig, og vi kan bruke den fra Excel. Vi går inn på Data og velger Fra andre kilder. Her velger vi Fra Microsoft Query:

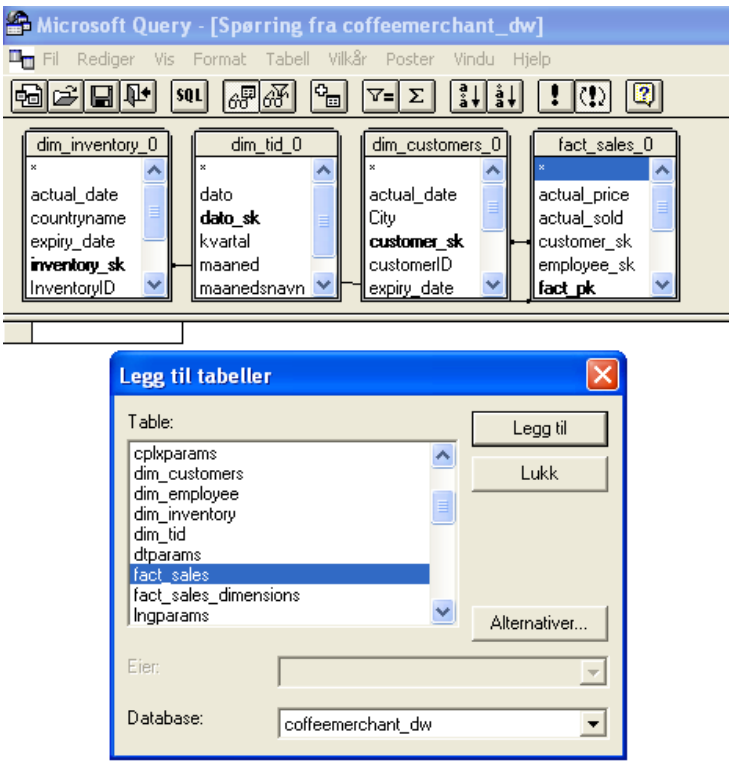


Vi får da opp en boks der vi ser de ODBC-koblingene vi har definert. Her har jeg valgt coffeemerchant_dw (som her er navnet på koblingen, ikke databasen):



Figur 15.28 Valg av ekstern database

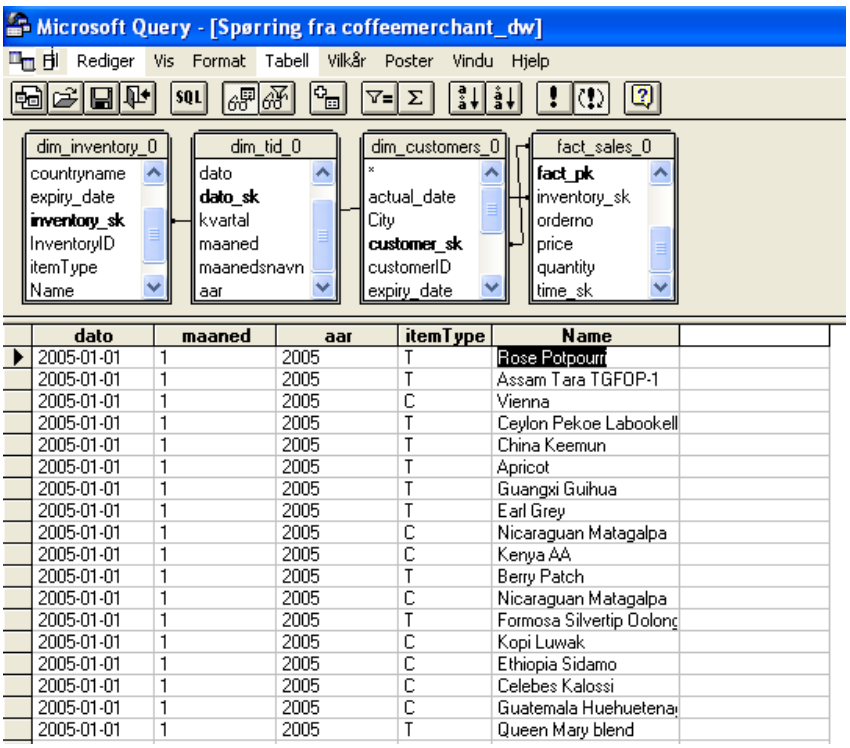
Dette gjør at Microsoft Query starter når vi klikker OK. Her er Query med fire tabeller lagt til (velg tabeller fra menylinjen og deretter Legg til):



Figur 15.29 Valgte tabeller vises i Query

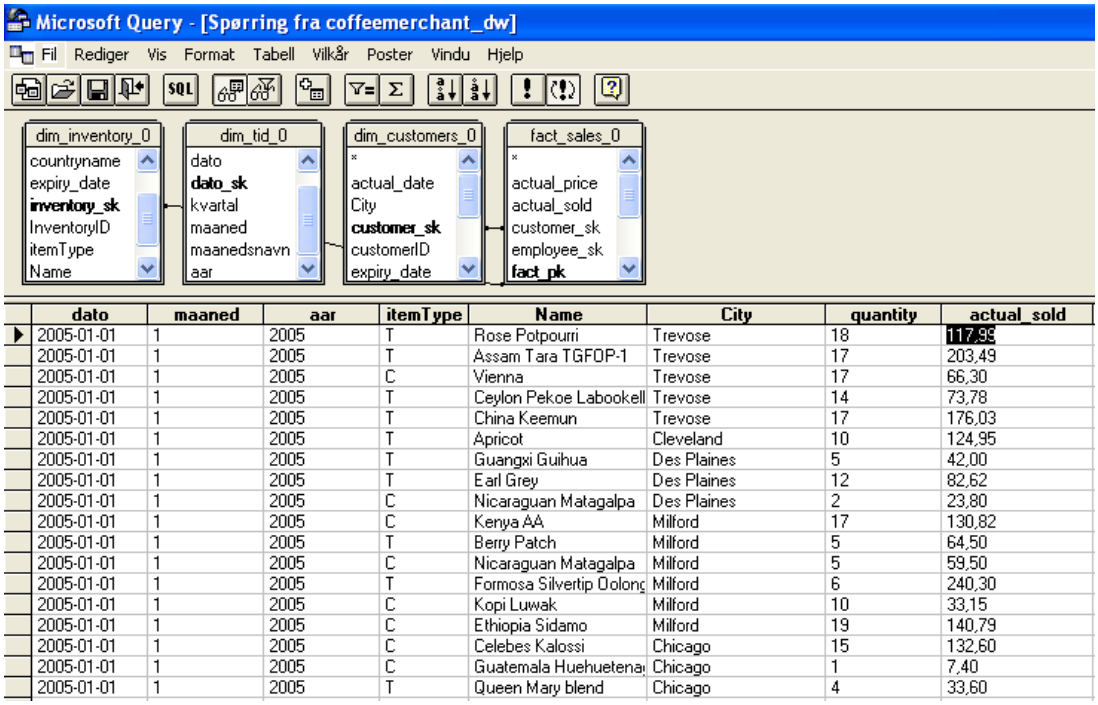
Query oppdager automatisk koblingene mellom tabellene ut fra fremmednøklene.

Vi lager nå en spørring rett og slett ved å dra feltene fra tabellene ned på arbeidsbordet. For hvert felt henter Query data for dette feltet:



Figur 15.30 Spørringen bygges opp i Query

Den ferdige spørringen ser slik ut:

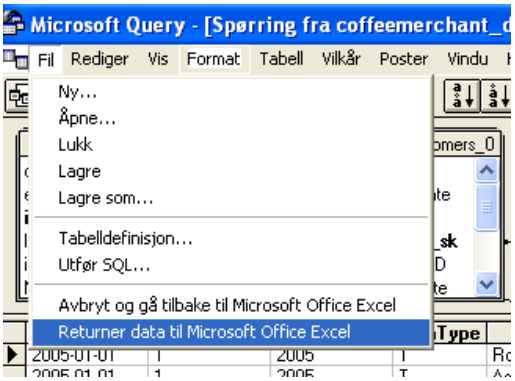


dato	maaned	aar	itemType	Name	City	quantity	actual_sold
2005-01-01	1	2005	T	Rose Potpourri	Trevose	18	117,99
2005-01-01	1	2005	T	Assam Tara TGFOP-1	Trevose	17	203,49
2005-01-01	1	2005	C	Vienna	Trevose	17	66,30
2005-01-01	1	2005	T	Ceylon Pekoe Labookell	Trevose	14	73,78
2005-01-01	1	2005	T	China Keemun	Trevose	17	176,03
2005-01-01	1	2005	T	Apricot	Cleveland	10	124,95
2005-01-01	1	2005	T	Guangxi Guihua	Des Plaines	5	42,00
2005-01-01	1	2005	T	Earl Grey	Des Plaines	12	82,62
2005-01-01	1	2005	C	Nicaraguan Matagalpa	Des Plaines	2	23,80
2005-01-01	1	2005	C	Kenya AA	Milford	17	130,82
2005-01-01	1	2005	T	Berry Patch	Milford	5	64,50
2005-01-01	1	2005	C	Nicaraguan Matagalpa	Milford	5	59,50
2005-01-01	1	2005	T	Formosa Silvertip Oolong	Milford	6	240,30
2005-01-01	1	2005	C	Kopi Luwak	Milford	10	33,15
2005-01-01	1	2005	C	Ethiopia Sidamo	Milford	19	140,79
2005-01-01	1	2005	C	Celebes Kalossi	Chicago	15	132,60
2005-01-01	1	2005	C	Guatemala Huehuetena	Chicago	1	7,40
2005-01-01	1	2005	T	Queen Mary blend	Chicago	4	33,60

Figur 15.31 Ferdig spørring

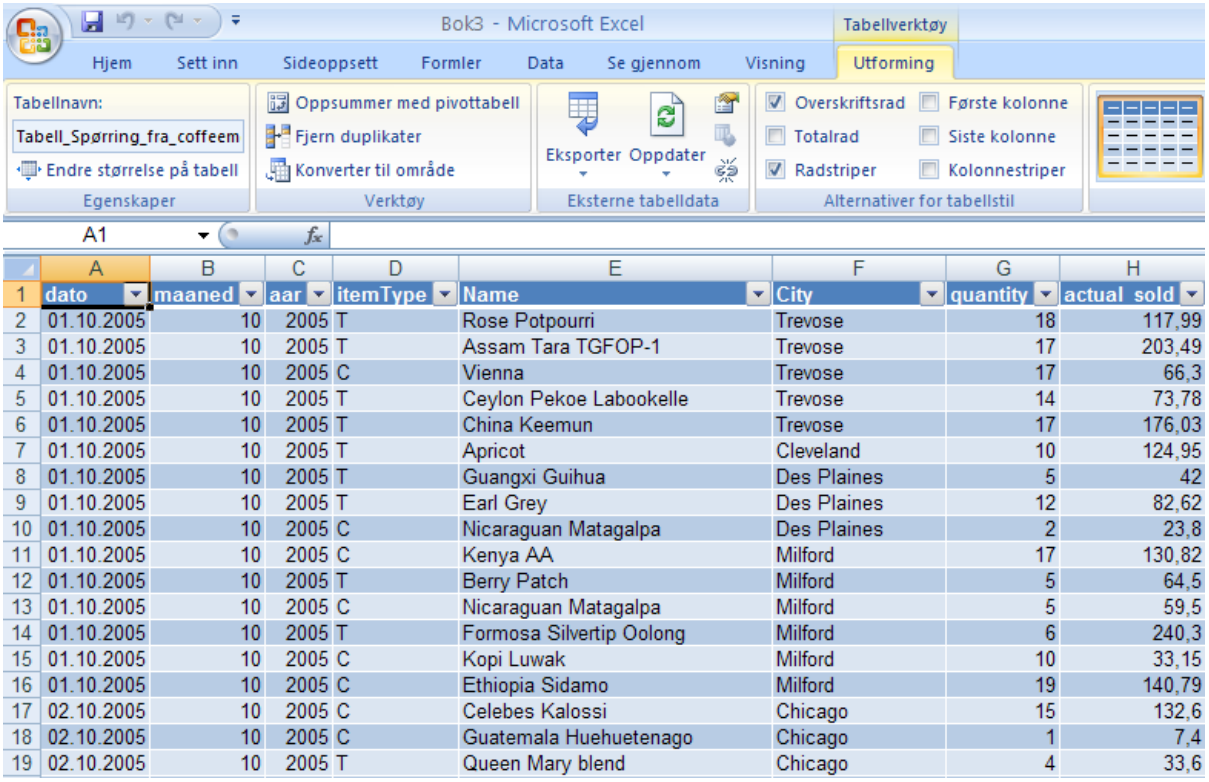
Vi kan nå lagre denne spørringen til senere bruk. Den kan da hentes inn når vi velger Fra Microsoft Query i regnearket ved at vi velger **Spørringer** i stedet for **Databaser**.

Etter at vi har lagret, velger vi File | Returner data til Microsoft Office Excel:



Figur 15.32 Import av data til Excel starter

Etter en tids jobbing, er resultatet av spørringen lagt inn i regnearket:



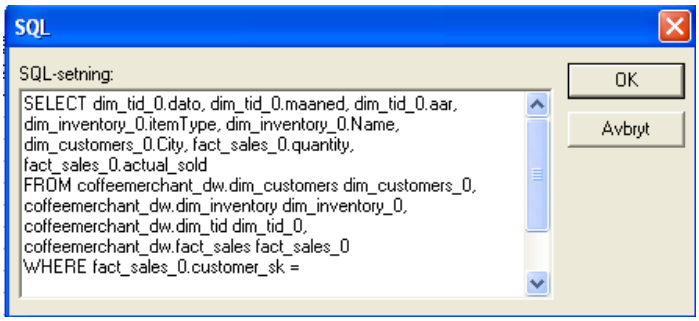
	A	B	C	D	E	F	G	H
1	dato	maaned	aar	itemType	Name	City	quantity	actual_sold
2	01.10.2005	10	2005	T	Rose Potpourri	Trevose	18	117,99
3	01.10.2005	10	2005	T	Assam Tara TGFOP-1	Trevose	17	203,49
4	01.10.2005	10	2005	C	Vienna	Trevose	17	66,3
5	01.10.2005	10	2005	T	Ceylon Pekoe Labookelle	Trevose	14	73,78
6	01.10.2005	10	2005	T	China Keemun	Trevose	17	176,03
7	01.10.2005	10	2005	T	Apricot	Cleveland	10	124,95
8	01.10.2005	10	2005	T	Guangxi Guihua	Des Plaines	5	42
9	01.10.2005	10	2005	T	Earl Grey	Des Plaines	12	82,62
10	01.10.2005	10	2005	C	Nicaraguan Matagalpa	Des Plaines	2	23,8
11	01.10.2005	10	2005	C	Kenya AA	Milford	17	130,82
12	01.10.2005	10	2005	T	Berry Patch	Milford	5	64,5
13	01.10.2005	10	2005	C	Nicaraguan Matagalpa	Milford	5	59,5
14	01.10.2005	10	2005	T	Formosa Silvertip Oolong	Milford	6	240,3
15	01.10.2005	10	2005	C	Kopi Luwak	Milford	10	33,15
16	01.10.2005	10	2005	C	Ethiopia Sidamo	Milford	19	140,79
17	02.10.2005	10	2005	C	Celebes Kalossi	Chicago	15	132,6
18	02.10.2005	10	2005	C	Guatemala Huehuetenago	Chicago	1	7,4
19	02.10.2005	10	2005	T	Queen Mary blend	Chicago	4	33,6

Figur 15.33 Ferdig import i Excel

Vi kan nå fortsette med å lage pivottabeller akkurat som før.

I dette tilfellet produserte faktisk spørringsveiviseren samme resultat, selv om Excel ga en melding om at den ikke kunne det. Veiviseren er først og fremst for å lage enkle spørringer, mens mer kompliserte spørringer må lages med Microsoft Query.

Det kan være lurt å sjekke den genererte SQL-setningen. Klikk på SQL-knappen i Query og sjekk at alt stemmer. I dette eksempelet hadde Query "glemt" å koble faktatabellen og tidsdimensjonen, antagelig fordi primærnøkkel og fremmednøkkel har forskjellig navn i de to tabellene. Resultatet av dette ble et kartesisk produkt. Vi ordner dette lett ved å skrive til de manglende delene av setningen.

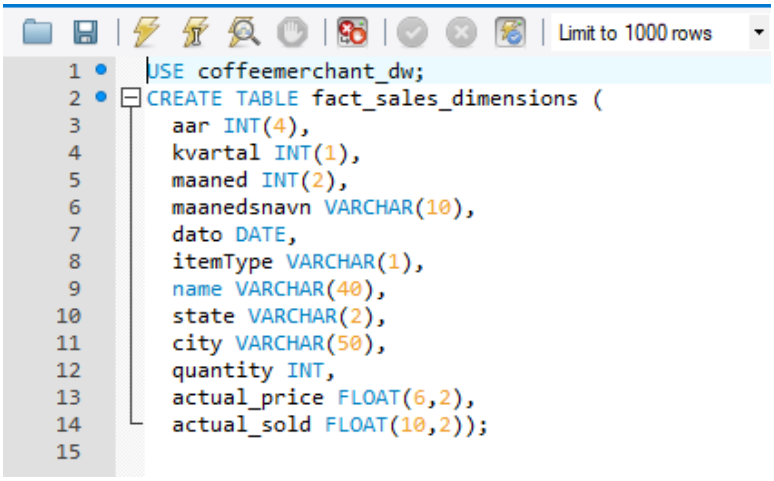


Figur 15.34 Kontroll av SQL-setning generert av Query

Query er et avansert program med mange muligheter. Den gjør det mulig å definere de mest avanserte spørringer for brukere uten SQL-kunnskaper.

15.3.6 Fra flat fil til regneark

Nedenfor er vist et script for å eksportere data fra coffeemerchant-datavarehuset til en tekstfil. Denne er litt annerledes enn den som er vist tidligere. Det er opprettet en tabell fact_sales_dimensions som er definert slik:



Scriptet for å fylle tabellen ser slik ut:

```
USE coffeemerchant_dw;
SELECT dim_tid.aar,dim_tid.kvartal,dim_tid.maaned,dim_tid.maanedsnavn,
dim_tid.dato,dim_inventory.itemType,dim_inventory.Name,dim_customers.state,dim_customers.city,
fact_sales.quantity,fact_sales.actual_price,fact_sales.actual_sold
FROM dim_tid,dim_inventory,dim_customers,fact_sales
WHERE fact_sales.time_sk = dim_tid.dato_sk
AND fact_sales.inventory_sk = dim_inventory.inventory_sk
AND fact_sales.consumer_sk = dim_customers.customer_sk
INTO OUTFILE 'C:/MySQL/coffeemerchant_salg.txt'
FIELDS ENCLOSED BY '"' TERMINATED BY ',' ESCAPED BY '"'
LINES TERMINATED BY '\r\n';
```

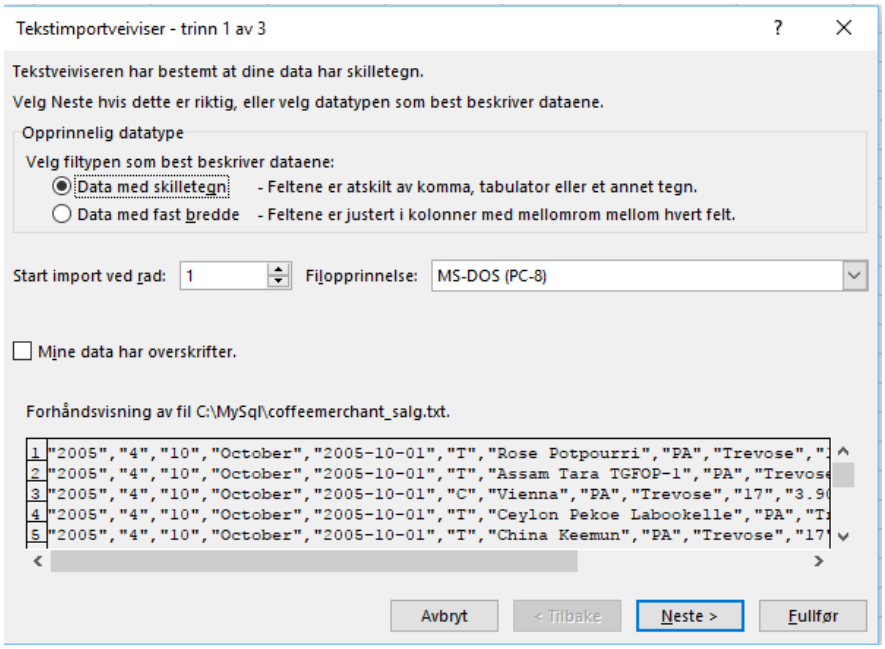
Figur 15.35 Script for å eksportere data til tekstfil

Den resulterende tekstfilen ser slik ut:

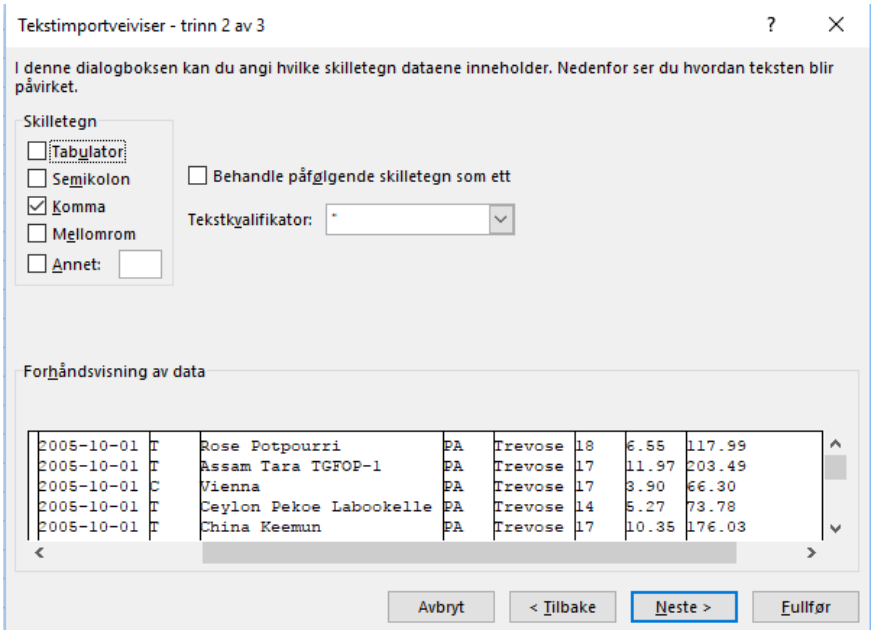
```
"2005","4","10","October","2005-10-01","T","Rose Potpourri","PA","Trevose","18","6.55","117.99"
"2005","4","10","October","2005-10-01","T","Assam Tara TGFOP-1","PA","Trevose","17","11.97","203.49"
"2005","4","10","October","2005-10-01","C","Vienna","PA","Trevose","17","3.90","66.30"
"2005","4","10","October","2005-10-01","T","Ceylon Pekoe Labookelle","PA","Trevose","14","5.27","73.78"
"2005","4","10","October","2005-10-01","T","China Keemun","PA","Trevose","17","10.35","176.03"
"2005","4","10","October","2005-10-01","C","Vanilla Nut Fudge","OH","Cleveland","19","5.03","95.67"
"2005","4","10","October","2005-10-01","C","Chocolate Hazelnut","OH","Cleveland","8","7.90","63.20"
"2005","4","10","October","2005-10-01","C","Chocolate Brandy","OH","Cleveland","15","5.95","89.25"
"2005","4","10","October","2005-10-01","C","Espresso Roast","OH","Cleveland","2","7.10","14.20"
"2005","4","10","October","2005-10-01","T","Apricot","OH","Cleveland","10","12.49","124.95"
"2005","4","10","October","2005-10-01","C","Chocolate Raspberry","IL","Des Plaines","17","4.50","76.50"
"2005","4","10","October","2005-10-01","T","Guangxi Guihua","IL","Des Plaines","5","8.40","42.00"
"2005","4","10","October","2005-10-01","C","Vanilla Nut Creme","IL","Des Plaines","14","4.51","63.07"
"2005","4","10","October","2005-10-01","T","Earl Grey","IL","Des Plaines","12","6.89","82.62"
"2005","4","10","October","2005-10-01","C","Nicaraguan Matagalpa","IL","Des Plaines","2","11.90","23.80"
"2005","4","10","October","2005-10-01","C","Cinnamon","MA","Milford","10","4.51","45.05"
"2005","4","10","October","2005-10-01","C","Kenya AA","MA","Milford","17","7.70","130.82"
"2005","4","10","October","2005-10-01","C","Vanilla Nut Creme","MA","Milford","14","3.83","53.55"
"2005","4","10","October","2005-10-01","T","Pan Fired Green","MA","Milford","19","6.84","129.96"
"2005","4","10","October","2005-10-01","T","Berry Patch","MA","Milford","5","12.90","64.50"
"2005","4","10","October","2005-10-01","C","Nicaraguan Matagalpa","MA","Milford","5","11.90","59.50"
"2005","4","10","October","2005-10-01","T","Formosa Silvertip Oolong","OH","Milford","6","40.05","240.30"
```

Figur 15.35 Tekstfilen eksportert fra fact_sales_dimensions

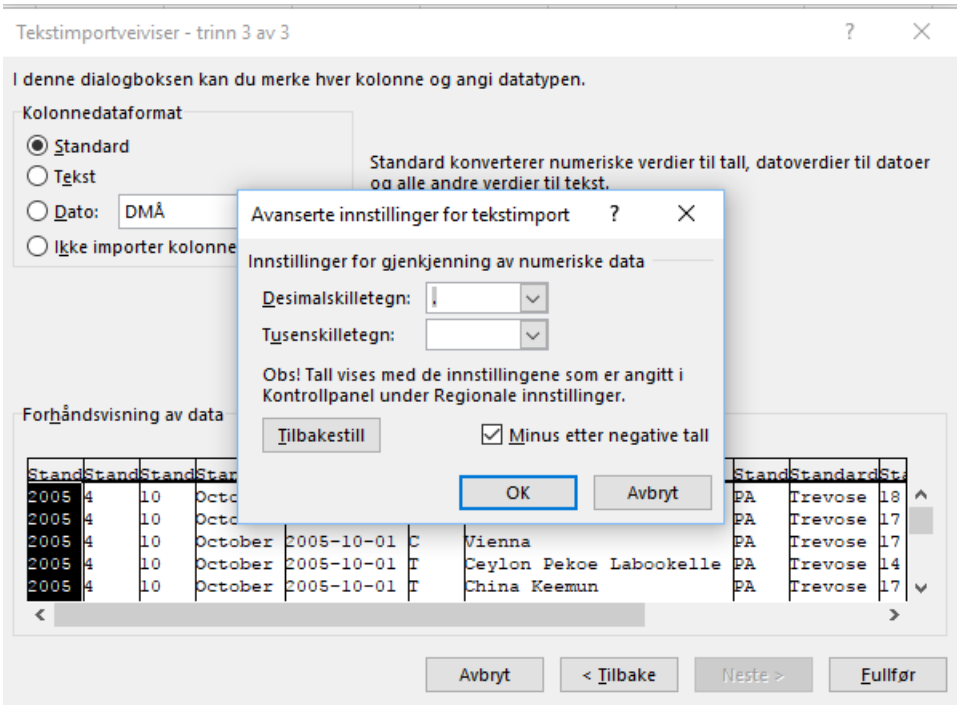
Denne filen kan vi nå åpne på vanlig måte i Excel. Programmet «forstår» at det er en tekstfil, og vil «spørre» hvordan den skal konverteres:



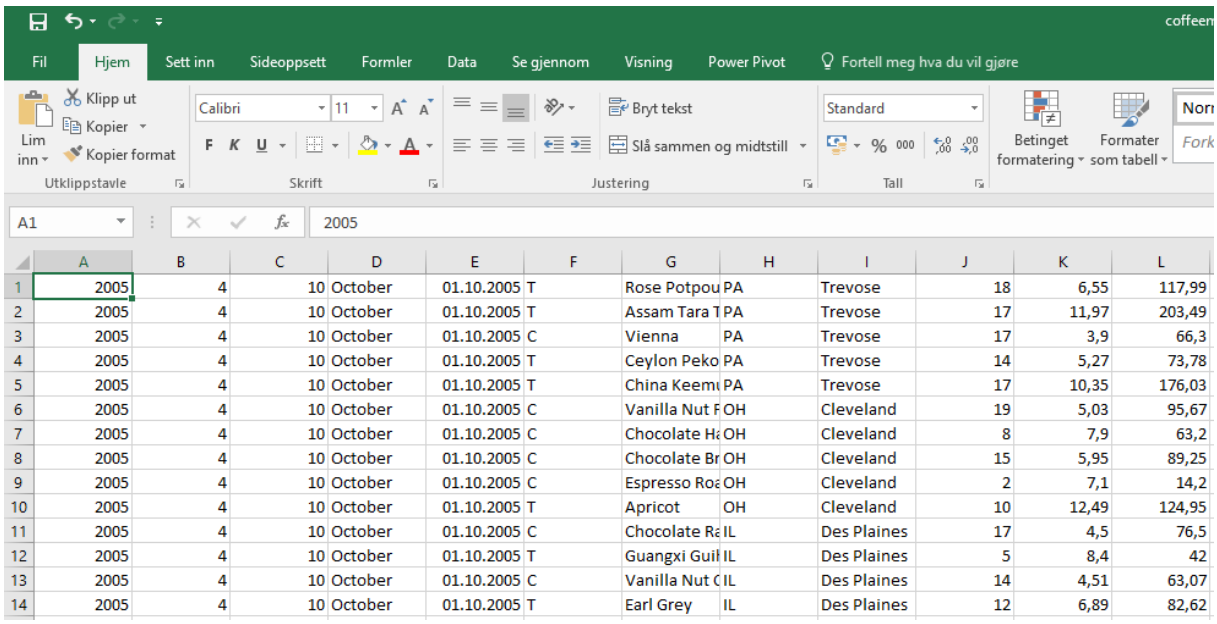
Vi klikker neste, og kan nå velge hva som er skilletegn. Vi velger komma, og vi presenteres for hvordan regnearket vil se ut:



Pass på å angi at desimaltegnet er punktum, ellers kan de skje merkelige ting med desimaltallene. Klikk Neste, og så Avansert:



Og i regnearket blir det da slik:



	A	B	C	D	E	F	G	H	I	J	K	L
1	2005	4	10	October	01.10.2005	T	Rose Potpou	PA	Trevose	18	6,55	117,99
2	2005	4	10	October	01.10.2005	T	Assam Tara T	PA	Trevose	17	11,97	203,49
3	2005	4	10	October	01.10.2005	C	Vienna	PA	Trevose	17	3,9	66,3
4	2005	4	10	October	01.10.2005	T	Ceylon Peko	PA	Trevose	14	5,27	73,78
5	2005	4	10	October	01.10.2005	T	China Keem	PA	Trevose	17	10,35	176,03
6	2005	4	10	October	01.10.2005	C	Vanilla Nut F	OH	Cleveland	19	5,03	95,67
7	2005	4	10	October	01.10.2005	C	Chocolate Ha	OH	Cleveland	8	7,9	63,2
8	2005	4	10	October	01.10.2005	C	Chocolate Br	OH	Cleveland	15	5,95	89,25
9	2005	4	10	October	01.10.2005	C	Espresso Roa	OH	Cleveland	2	7,1	14,2
10	2005	4	10	October	01.10.2005	T	Apricot	OH	Cleveland	10	12,49	124,95
11	2005	4	10	October	01.10.2005	C	Chocolate Ra	IL	Des Plaines	17	4,5	76,5
12	2005	4	10	October	01.10.2005	T	Guangxi Guil	IL	Des Plaines	5	8,4	42
13	2005	4	10	October	01.10.2005	C	Vanilla Nut C	IL	Des Plaines	14	4,51	63,07
14	2005	4	10	October	01.10.2005	T	Earl Grey	IL	Des Plaines	12	6,89	82,62

Før vi kan lage pivottabeller, må vi legge til kolonneoverskrifter. Det er i den første raden Excel ser etter hvilke felt som skal være med i en pivottabell:

	A	B	C	D	E	F	G	H	I	J	K	L
1	år	kvartal	månedsnr	måned	dato	varegruppe	vare	stat	by	antall	pris	salg
2	2005	4	10	October	01.10.2005	T	Rose Potpourri	PA	Trevose	18	6,55	117,99
3	2005	4	10	October	01.10.2005	T	Assam Tara TGFOP-1	PA	Trevose	17	11,97	203,49
4	2005	4	10	October	01.10.2005	C	Vienna	PA	Trevose	17	3,9	66,3
5	2005	4	10	October	01.10.2005	T	Ceylon Pekoe Labookelle	PA	Trevose	14	5,27	73,78
6	2005	4	10	October	01.10.2005	T	China Keemun	PA	Trevose	17	10,35	176,03
7	2005	4	10	October	01.10.2005	C	Vanilla Nut Fudge	OH	Cleveland	19	5,03	95,67
8	2005	4	10	October	01.10.2005	C	Chocolate Hazelnut	OH	Cleveland	8	7,9	63,2
9	2005	4	10	October	01.10.2005	C	Chocolate Brandy	OH	Cleveland	15	5,95	89,25
10	2005	4	10	October	01.10.2005	C	Espresso Roast	OH	Cleveland	2	7,1	14,2
11	2005	4	10	October	01.10.2005	T	Apricot	OH	Cleveland	10	12,49	124,95
12	2005	4	10	October	01.10.2005	C	Chocolate Raspberry	IL	Des Plaines	17	4,5	76,5
13	2005	4	10	October	01.10.2005	T	Guangxi Guihua	IL	Des Plaines	5	8,4	42
14	2005	4	10	October	01.10.2005	C	Vanilla Nut Creme	IL	Des Plaines	14	4,51	63,07
15	2005	4	10	October	01.10.2005	T	Earl Grey	IL	Des Plaines	12	6,89	82,62
16	2005	4	10	October	01.10.2005	C	Nicaraguan Matagalpa	IL	Des Plaines	2	11,9	23,8
17	2005	4	10	October	01.10.2005	C	Cinnamon	MA	Milford	10	4,51	45,05

Nå er det bare å begynne å lage pivottabeller!

Kapittel 16 Microsoft Power BI

Power BI er Microsofts verktøy for Business Intelligence analyser og visualisering. Det kommer i to former, som også kan brukes om hverandre: Power BI Service, som er skybasert, og Power BI Desktop. Det er den siste vi skal bruke.

16.1 Hva er Power BI?

Power BI er laget for at folk som driver med analyse av data skal kunne gjøre det på egen hånd, uten å være avhengig av IT-ekspertise. Power BI er laget først og fremst på grunnlag av følgende programvare:

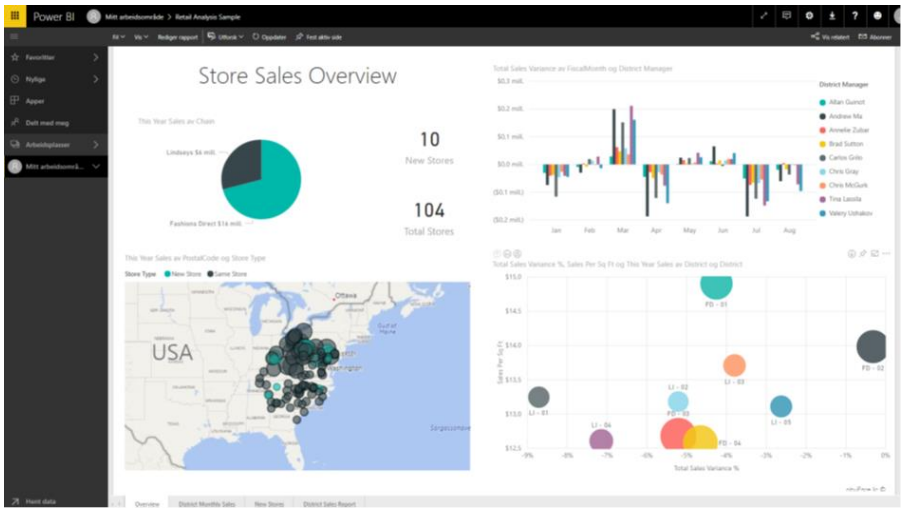
- Power Query: Et verktøy for innhenting og transformasjon av data. Kan lese data fra en rekke databasesystemer, CSV, regnearkfiler, Facebook m.m.
- Power Pivot: Et verktøy for å modellere tabelldata. Bruker et språk som kalles DAX (Data Analysis eXpression language for summeringer og kalkyler.
- Power View: Et verktøy for visualisering av data. Data kan vises i tabeller og grafer, man kan filtrere, drille, slice og dice.

Disse verktøyene har opprinnelig blitt utviklet som plug-ins til Excel regneark. Pover Pivot kunne lastes ned som plug-in til Excel 2010 og 2013, og er innebygget i Excel 2016.

Power BI samler de tidligere plug-ins til Excel, og har også et par tillegg:

- Power Map: Et verktøy for å visualisere data i 3D-kart.
- Power Q&A: Et verktøy for å stille spørsmål i naturlig språk.

I utgangspunktet finer vi disse delene i den skybaserte Power BI Service. Alt dette finnes også i utviklingsverktøyet Power BI Desktop. Tilgang til Power BI Service er foreløpig en del av Office 365, men det sies at dette ikke vil være tilfelle i fremtiden.

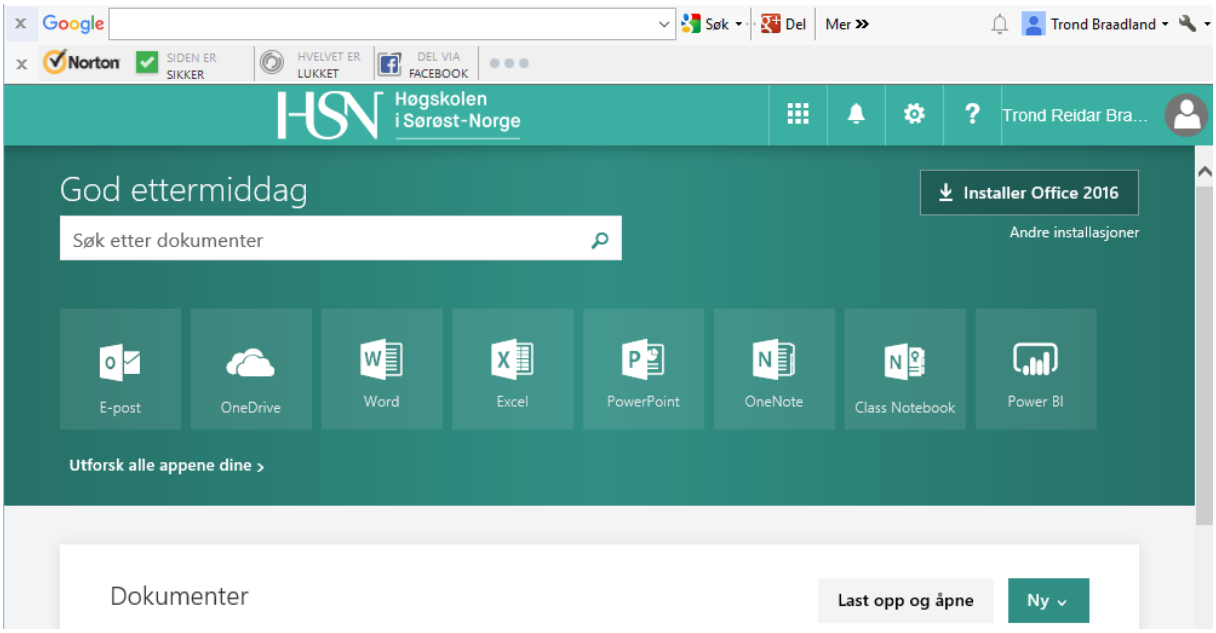


Figur 16.1 Visualiseringseksempel på Power BI Service

Power BI Desktop, som vi skal bruke, er en desktop-applikasjon som samler Power Query, Pover Pivot og Power View i en integrert enhet. Power BI Desktop oppdateres regelmessig.

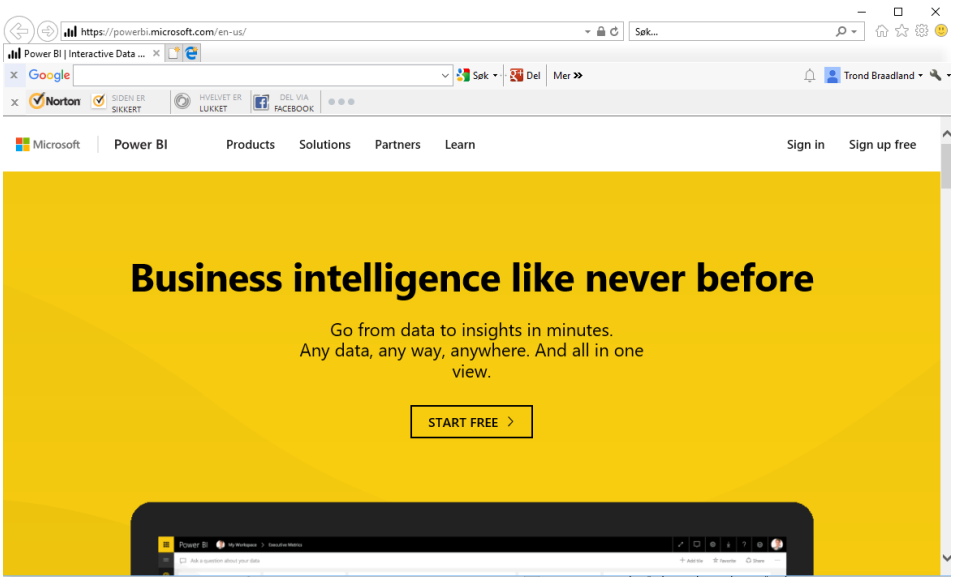
16.2 Power BI server

Vi skal først se litt på Power BI Server. Vi kan gå inn på hjemmesiden https://powerbi.microsoft.com/en-us/ . Har vi Office 365m inngår Power BI inngår i denne. Her er vist Office 365-menyen slik den vises ved Høgskolen i Sørøst-Norge:



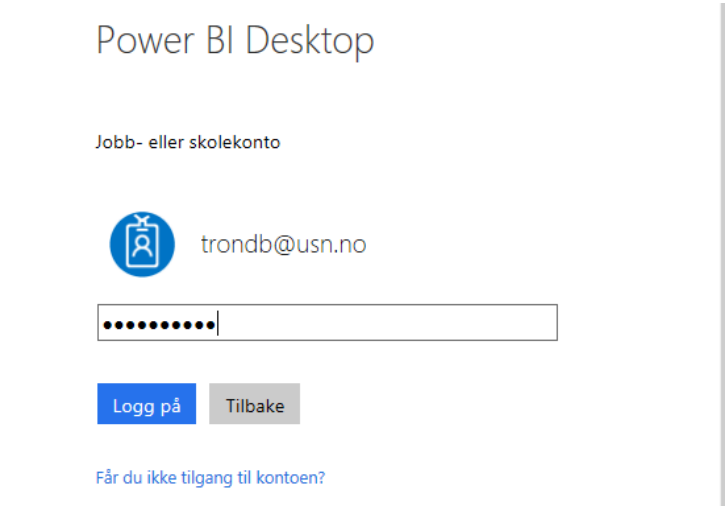
Figur 16.2 Office 365 menyen

Velkomstsiden til Power BI er vist nedenfor:



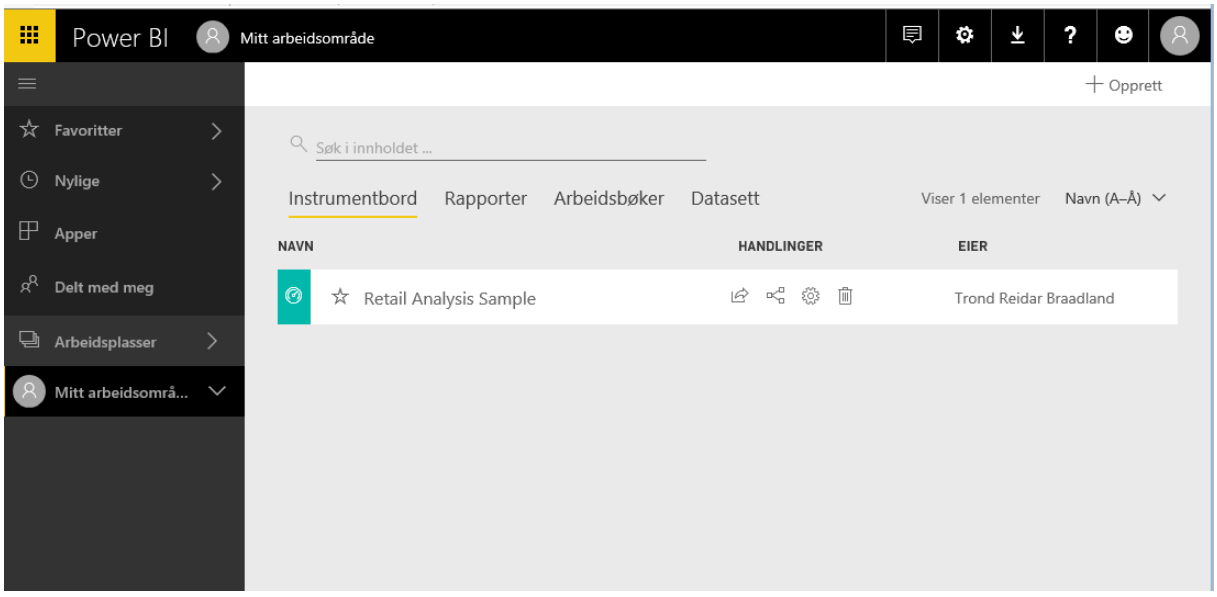
Figur 16.3 Velkommen til Power BI

Du kan nå logge inn med din Microsoft ID. Dette vil være din usn-epostadresse. Pass på at du ikke kan bruke aliaset (i mitt tilfelle trond.braadland@usn.no), men må bruke det «egentlige» kontonavnet.



Figur 16.4 Innlogging med skolekonto

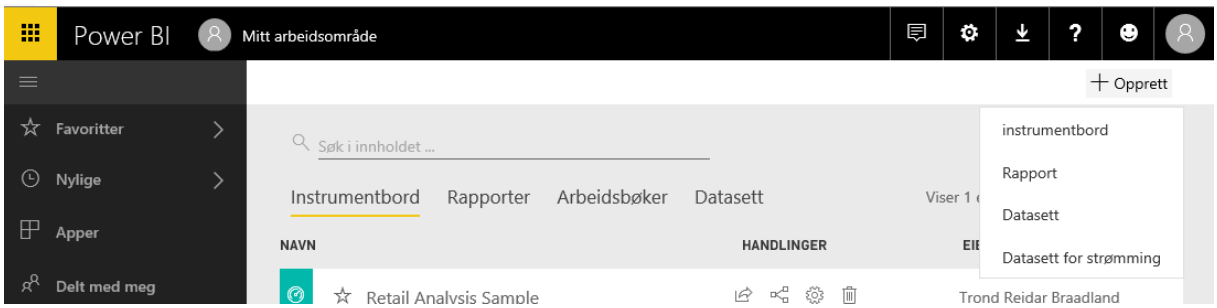
Her er Power BI Server startet, og en demo er lagt inn i Mitt arbeidsområde. Legg merke til at etter innlogging er menyene på norsk.



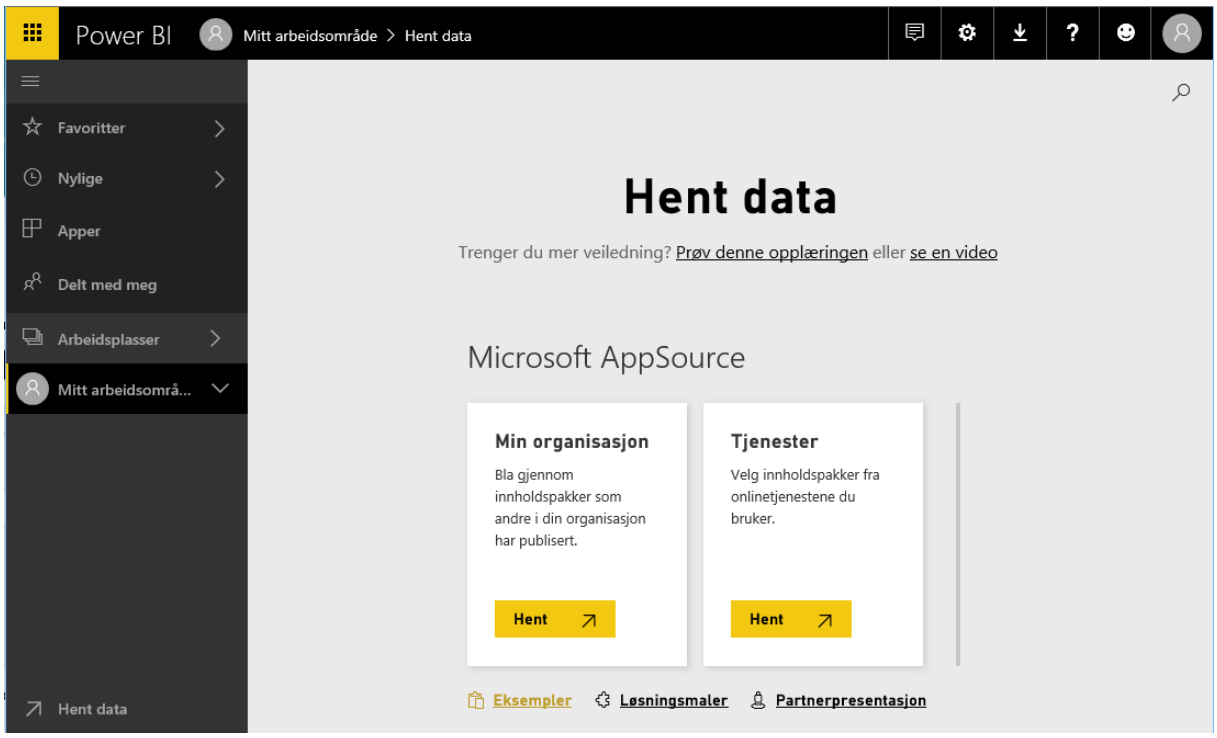
Figur 16.5 Arbeidsområdet til BI Server med importert datasett

Vi kan hente inn flere demoer ved å klikke på + Opprett i øverste høyre hjørne:

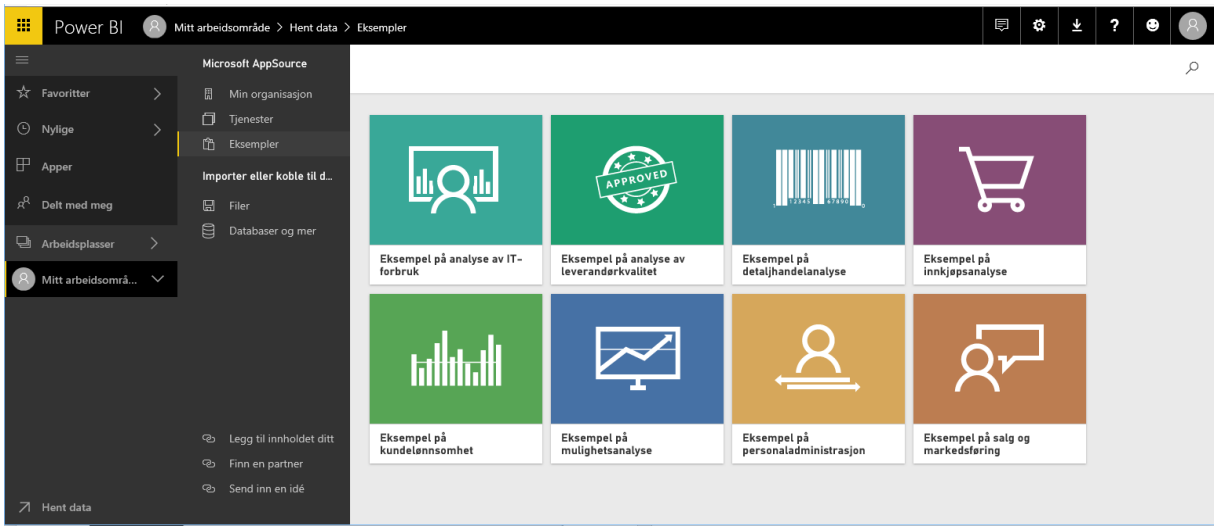
Velg Datasett:



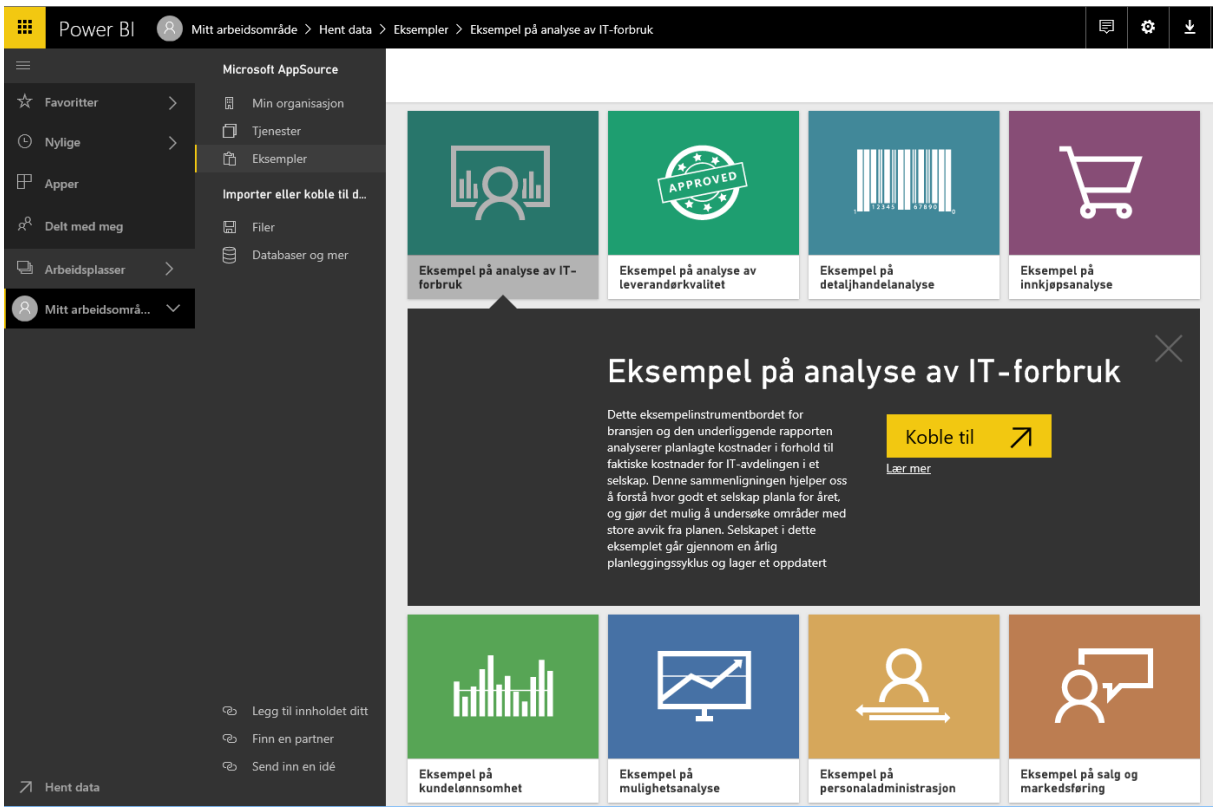
Velg så Eksempler nederst på siden:



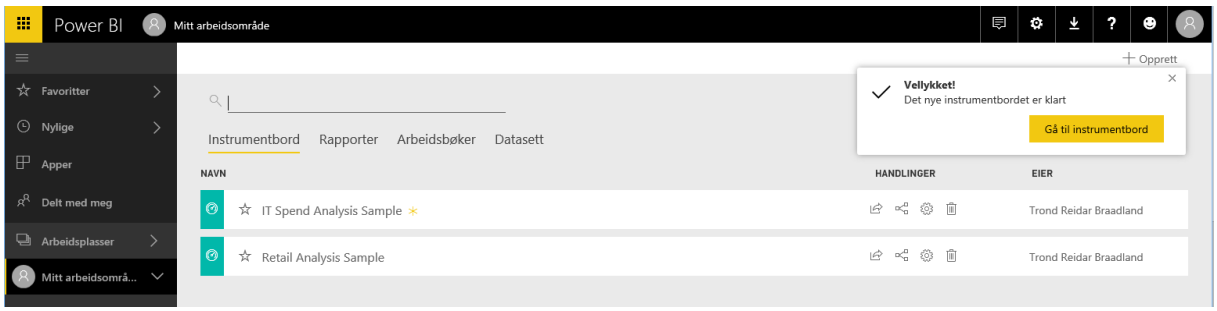
Vi får opp en mosaikk med de demodatasettene som er tilgjengelig:



Vi velger «Eksempel på analyse av IT-forbruk». Vi blir så spurt om å koble til datasettet:



Tilkoblingen var vellykket, og det nye datasettet ligger i Mitt arbeidsområde:

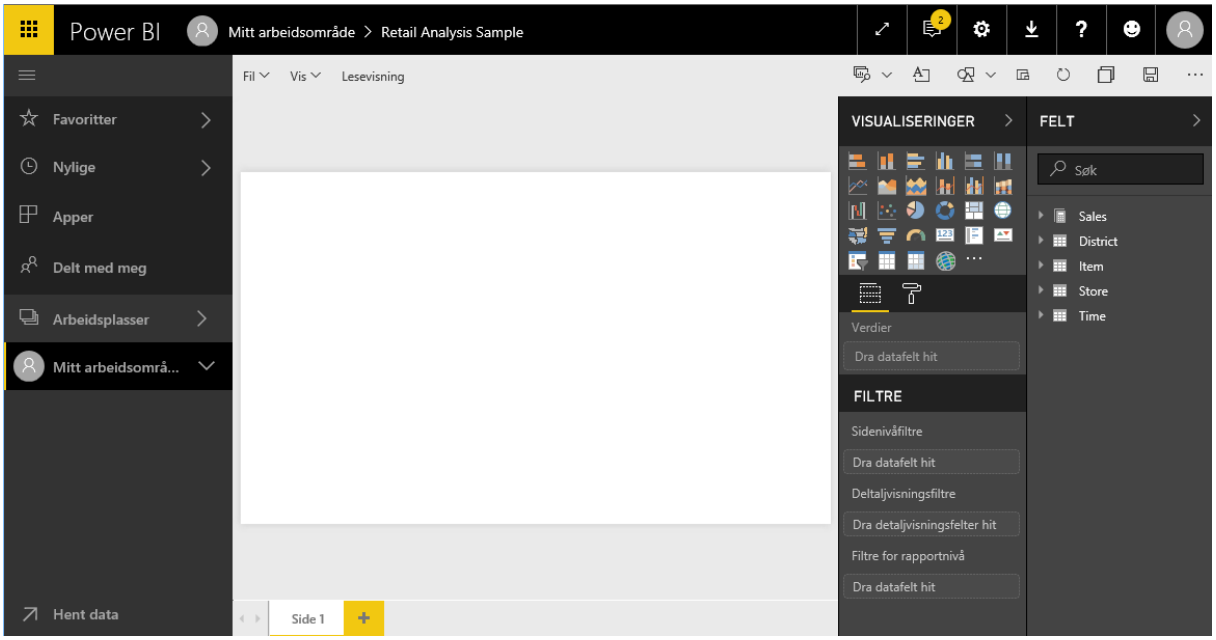


Opprette ny rapport fra et datasett (klikk på symbol i rød sirkel):

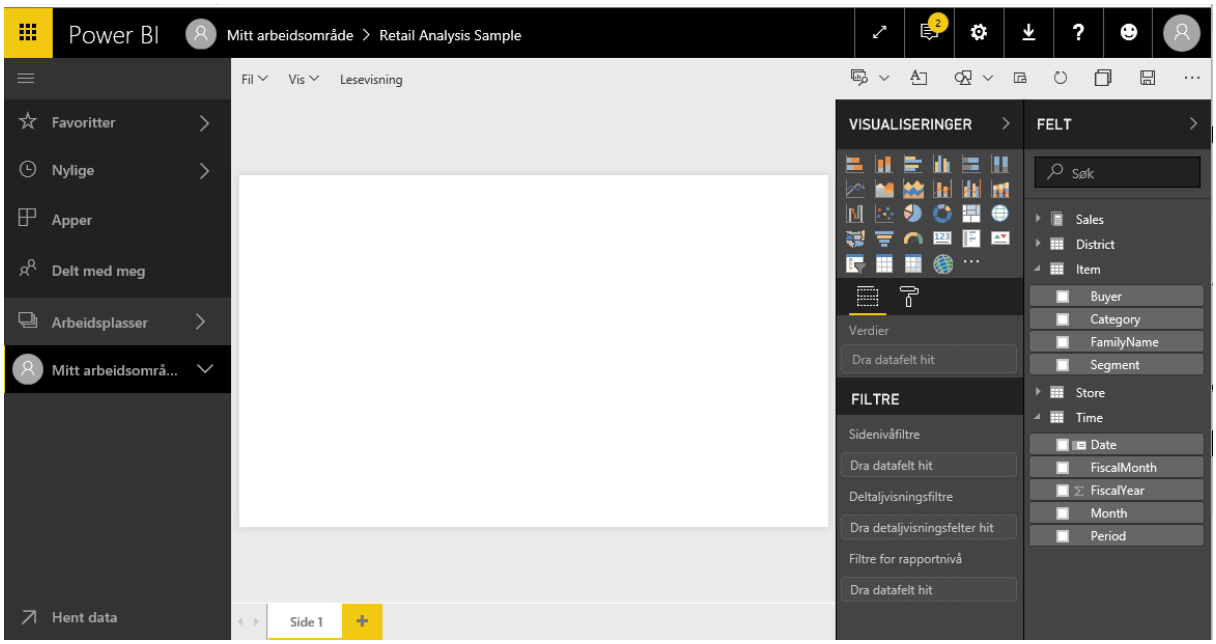
Vi kan nå designe en ny rapport. Når vi klikker på rapportsymbolet, får vi opp et vindu til høyre:

Vi skal først lage en rapport i Power BI. Under Datasett finner vi denne linjen:

Klikk på diagramsymbolet som her er ringet inn med rødt. Vi får da se tabellene i datasettet, og kan begynne å designe rapporten:



Klikker vi på en tabell, får vi se kolonnene den inneholder:



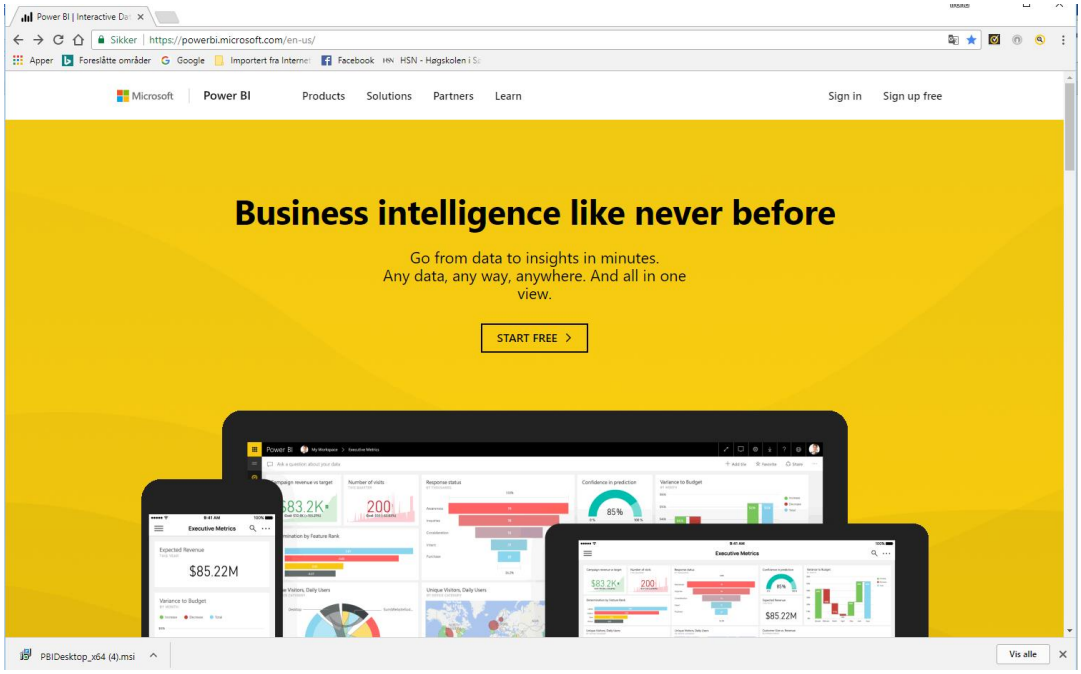
Vi skal imidlertid ikke bruke server-utgaven av Power BI. I stedet skal vi bruke en applikasjon for frittstående maskin, Power BI Desktop.

Kapittel 17 Power BI Desktop

Power BI Desktop er, som navnet sier, desktoputgaven av Power BI. Mens serverutgaven først og fremst er ment for sluttbrukere, er desktoputgaven ment for utviklere.

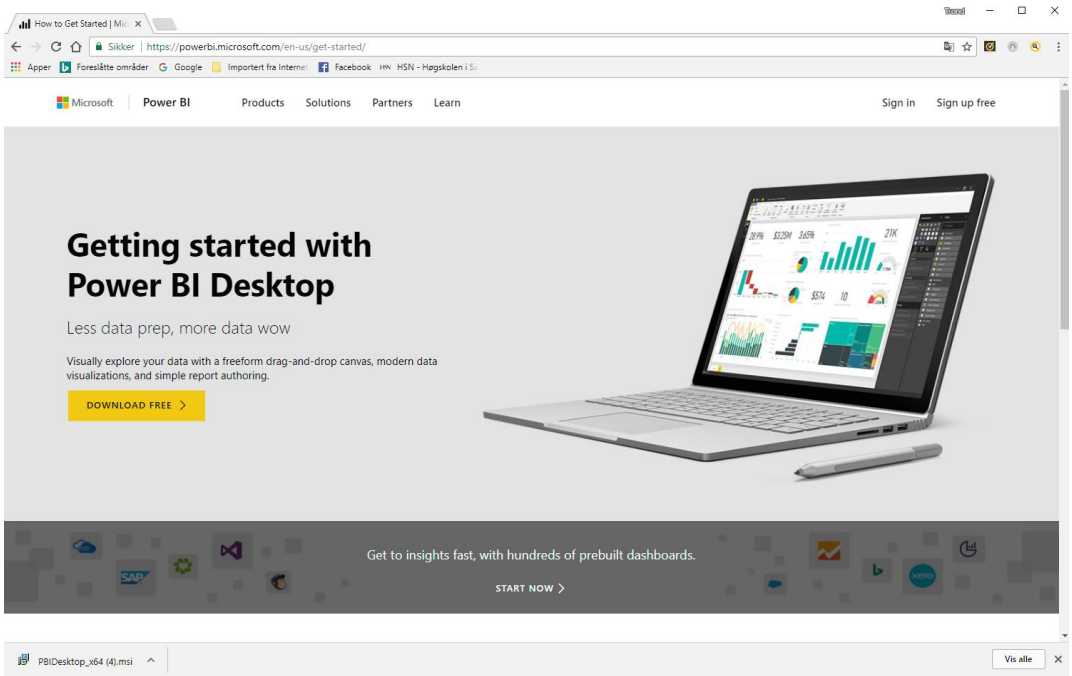
17.1 Nedlasting og installasjon av PowerBI

Gå til nettstedet til PowerBI: https://powerbi.microsoft.com/en-us/



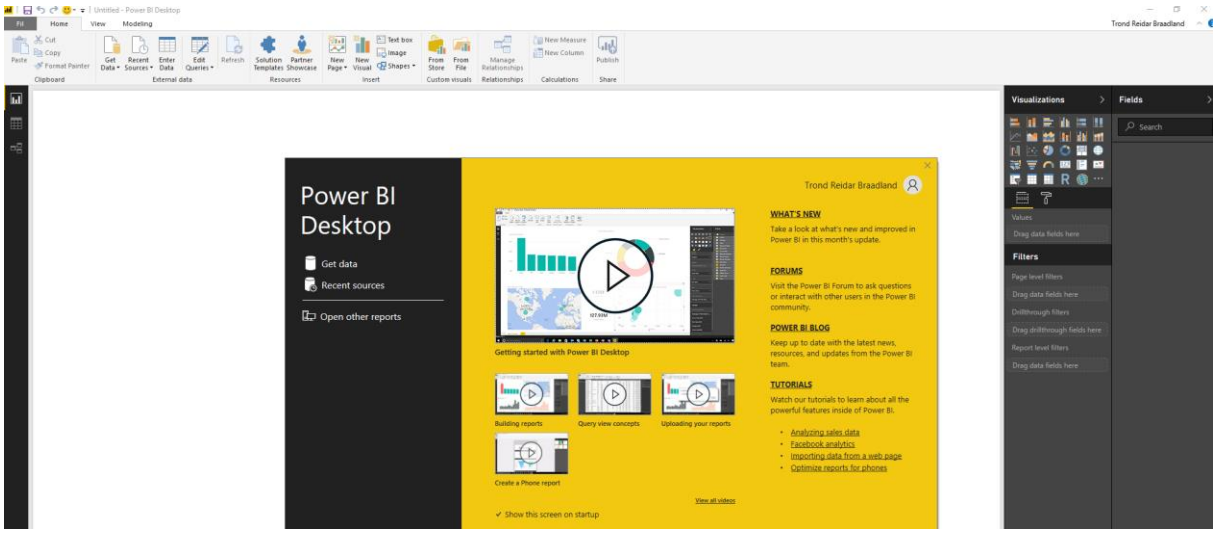
Figur 17.1 Startside hos Power BI

Klikk på START FREE, og du kan laste ned PowerBI Desktop:



Figur 17.2 Klar til nedlasting

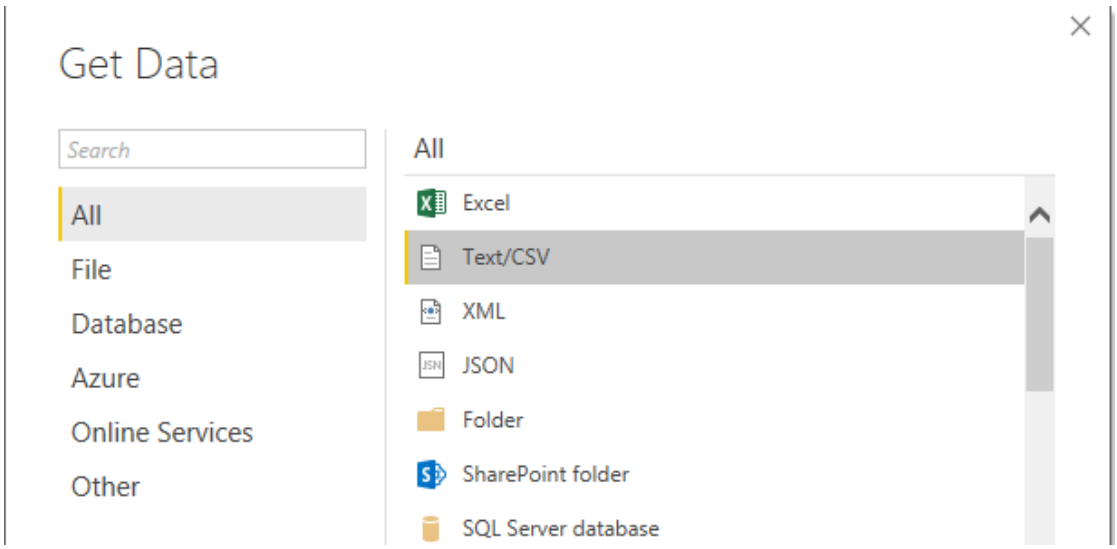
Kjør installasjonsfilen, og start PowerBI. I oktober 2017 ser PowerBI Desktop slik ut (dette kan endres med neste oppdatering):



Figur 17.3 Oppstart av Power BI Desktop

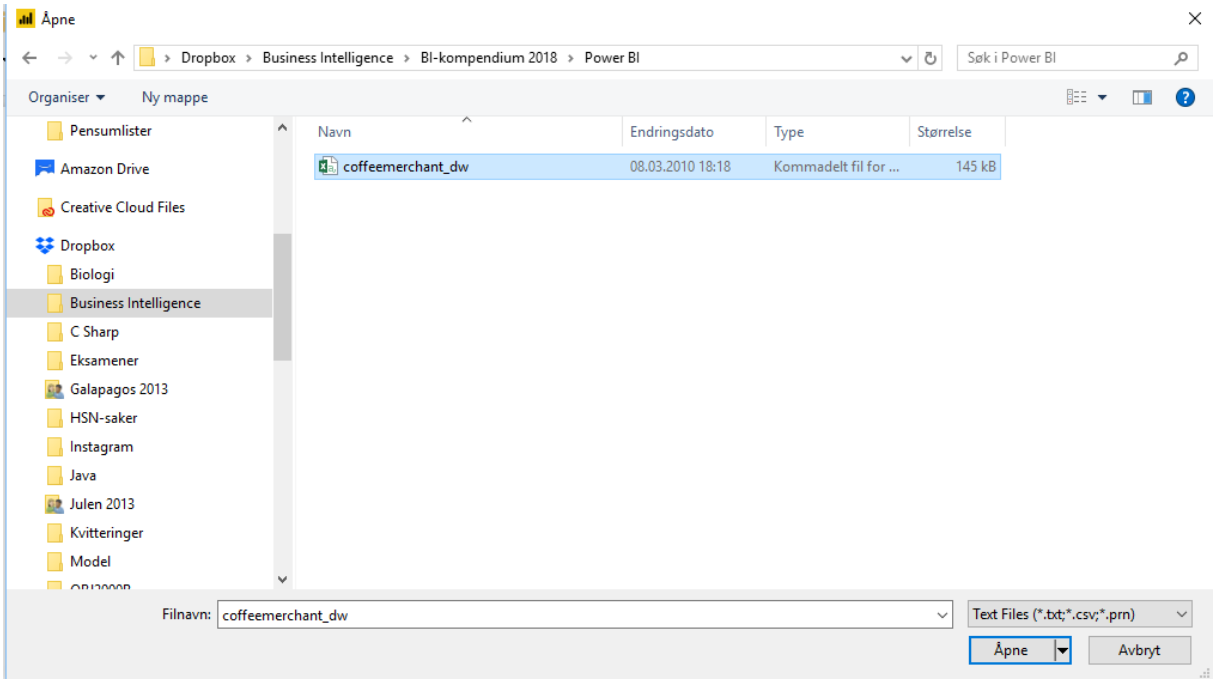
17.2 Import av CSV-fil

Vi skal importere den kommaseparerte filen med data fra Coffeemerchant-datavarehuset. Siden dette er et standard format for utveksling av data, en CSV-fil, angir vi for PowerBI at det er dette vi skal importere:



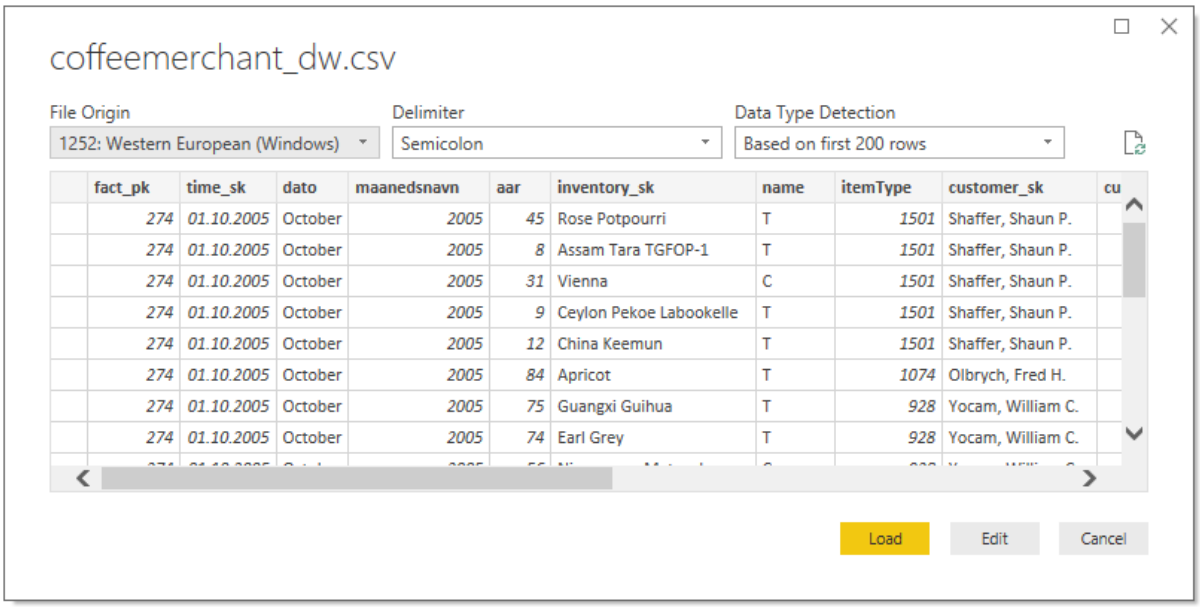
Figur 17.4 Valg av import av CSV-fil

Vi leter oss frem til filen:



Figur 17.5 Filen er laget ved eksport fra en database

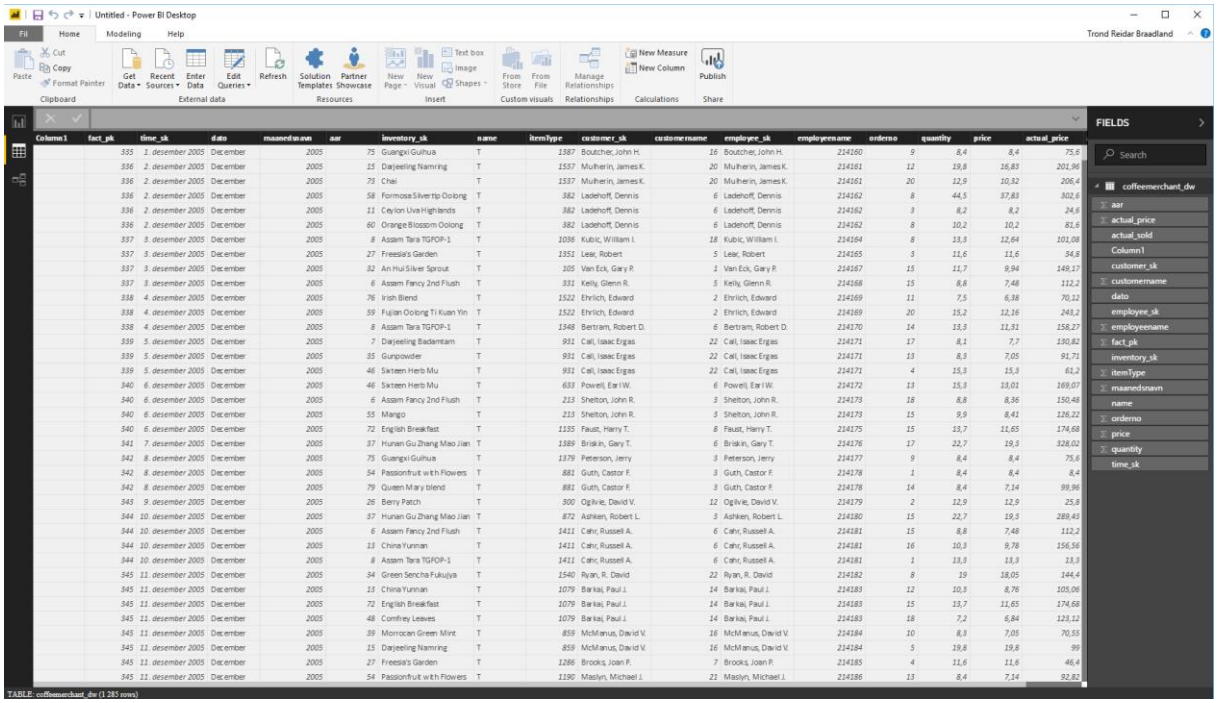
Når vi klikker Åpne, får vi et preview av innholdet:



fact_pk	time_sk	dato	maanedsnavn	aar	inventory_sk	name	itemType	customer_sk	cu
274	01.10.2005	October	2005	45	Rose Potpourri	T	1501	Shaffer, Shaun P.	
274	01.10.2005	October	2005	8	Assam Tara TGFOP-1	T	1501	Shaffer, Shaun P.	
274	01.10.2005	October	2005	31	Vienna	C	1501	Shaffer, Shaun P.	
274	01.10.2005	October	2005	9	Ceylon Pekoe Labookelle	T	1501	Shaffer, Shaun P.	
274	01.10.2005	October	2005	12	China Keemun	T	1501	Shaffer, Shaun P.	
274	01.10.2005	October	2005	84	Apricot	T	1074	Olbrych, Fred H.	
274	01.10.2005	October	2005	75	Guangxi Guihua	T	928	Yocam, William C.	
274	01.10.2005	October	2005	74	Earl Grey	T	928	Yocam, William C.	

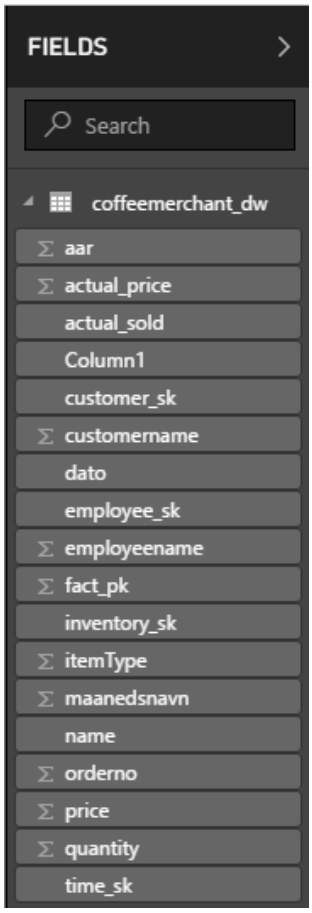
Figur 17.6 Preview av filen

Ser dette bra ut, trykker vi på Load og data hentes inn:



Figur 17.7 Filen er importert

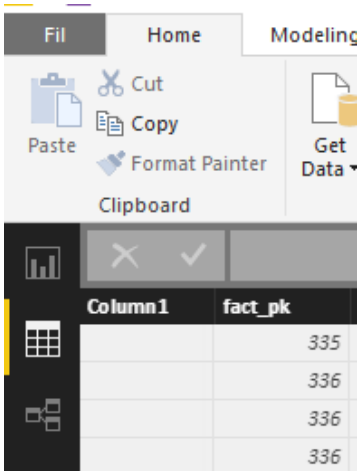
Før vi begynner å lage en rapport, må vi endre på datatypene til noen av kolonnene. Se på kolonnene slik de presenteres helt til høyre:



Figur 17.8 Alle kolonnene i den importerte tabellen

Summetegnene foran et kolonnenavn viser at Power BI antar at dette er tall som skal summeres. Vi må forandre datatypen først.

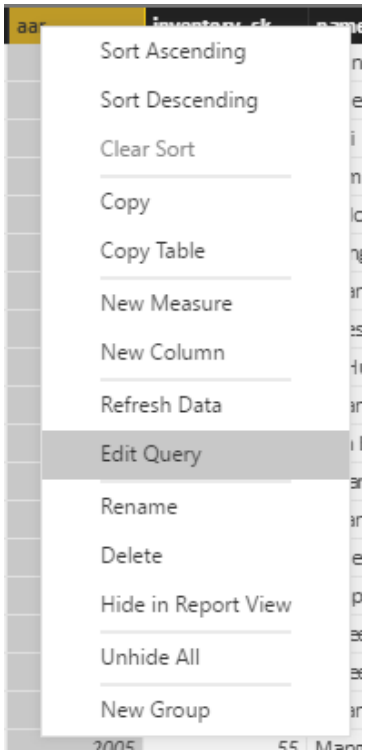
Øverst til venstre i Power BI ser vi disse tre ikonene:



Figur 17.9 Ikoner for de tre perspektivene i Power BI

Disse bruker vi til å veksle mellom presentasjon (øverst), data (i midten) og relasjoner.

Når vi er i datavisning, høyreklikke vi på feltet og velger Edit Query:



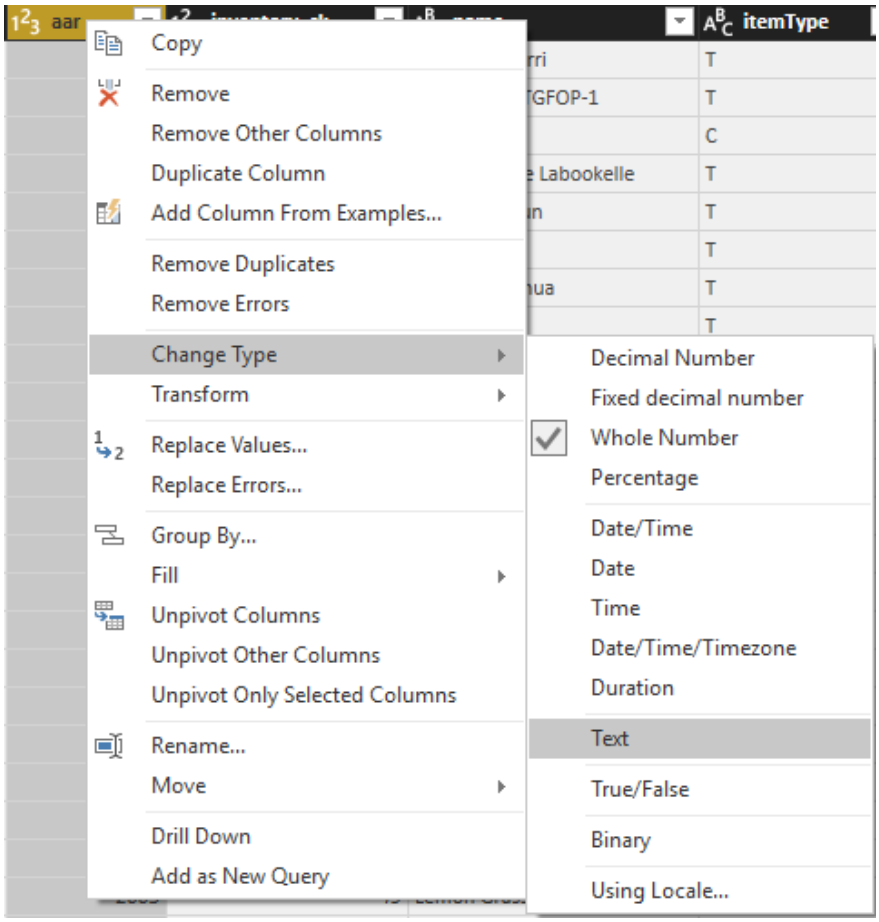
Figur 17.10 Klar til redigering

I query-vinduet som kommer opp, ser vi datatypene for de forskjellige kolonnene:

time_sk	dato	maanedsnavn	aar	inventory_sk	name	itemType	customer_sk
274	01.10.2005	October	2005	45	Rose Potpourri	T	
274	01.10.2005	October	2005	8	Assam Tara TGFOP-1	T	
274	01.10.2005	October	2005	31	Vienna	C	
274	01.10.2005	October	2005	9	Ceylon Pekoe Labookelle	T	
274	01.10.2005	October	2005	12	China Keemun	T	
274	01.10.2005	October	2005	84	Apricot	T	
274	01.10.2005	October	2005	75	Guangxi Guihua	T	
274	01.10.2005	October	2005	74	Earl Grey	T	

Figur 17.11 Query-vindu med datatyper

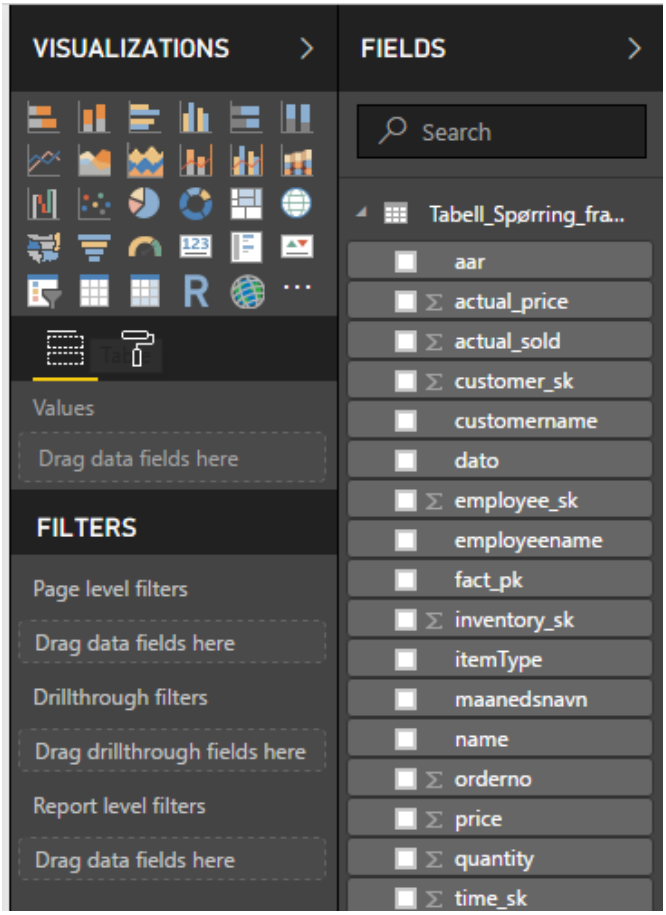
Symbolet foran månedsnavn viser at dette oppfatter Power BI som tekst (hvilket er korrekt), mens symbolet foran aar viser at Power BI tror dette er et tallfelt som skal summeres, hvilket ikke er korrekt. Alle felt som vi ikke ønsker skal summeres, redefinerer vi til tekst. Vi høyreklikker på aar, velger Change Type, og setter type til tekst:



Figur 17.12 Datatype for aar settes til Text

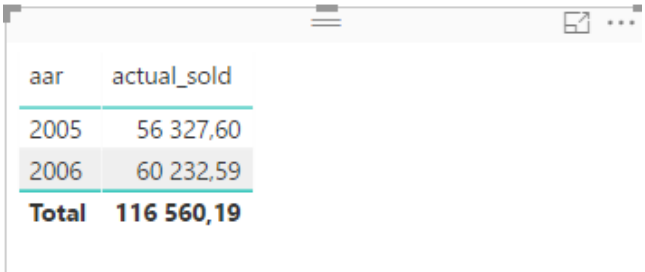
Nå er vi klar til å lage en rapport i form av en tabell.

Vi går til visualiseringsvinduet og velger en tabell (foreløpig har vi bare 1 tabell, nemlig den vi importerte):



Figur 17.13 Verktøykassen i Power BI

Tabellsymbolet er nummer to fra venstre i nederste rad (øvre venstre flate). Vi klikker nå i boksene foran aar og actual_sold. Merk at det står et summetegn (∑) foran actual_sold, det betyr at det blir summert opp på dette feltet. Resultatet blir dette:



aar	actual_sold
2005	56 327,60
2006	60 232,59
Total	**116 560,19**

Figur 17.14 Vår første rapport!

Vi kan legge til maanedsnavn:

aar	maanedsnavn	actual_sold
2005	December	20 559,75
2005	November	19 101,82
2005	October	16 666,03
2006	April	9 307,44
2006	February	15 898,63
2006	January	19 248,86
2006	March	15 777,66
Total		**116 560,19**

Figur 17.15 Her er det krysset av for maanedsnavn i tillegg til aar

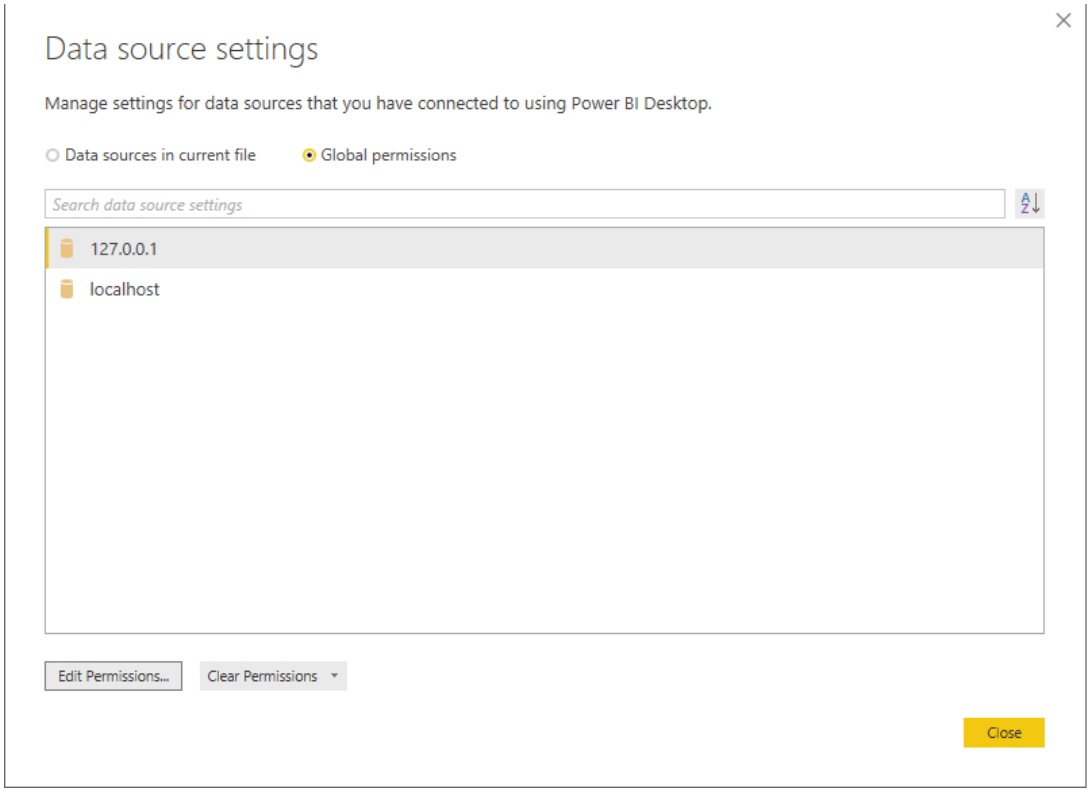
Power BI lager automatisk en spørring med GROUP BY.

17.3 Import fra MySQL

Nå er vi klar til å lage vår første analyse av MySQL-databasen.

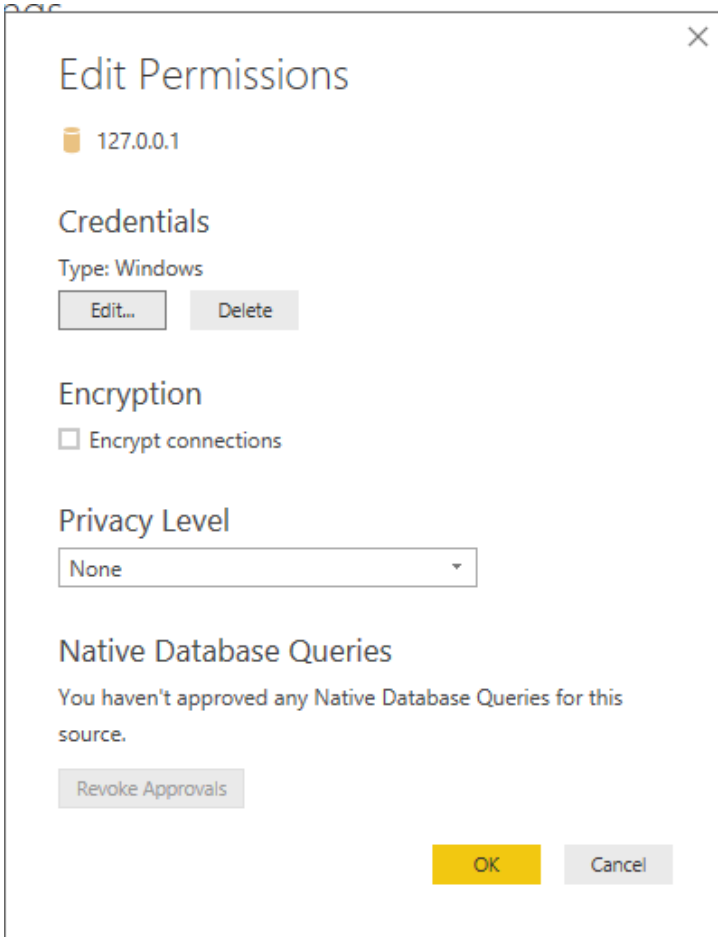
Først må vi installere MySQL Connector/net driveren:
https://dev.mysql.com/downloads/connector/net/

Deretter må vi klargjøre Power BI Desktop for oppkobling til MySQL. Gå inn på *File | Options and setting* og du får opp dette bildet:



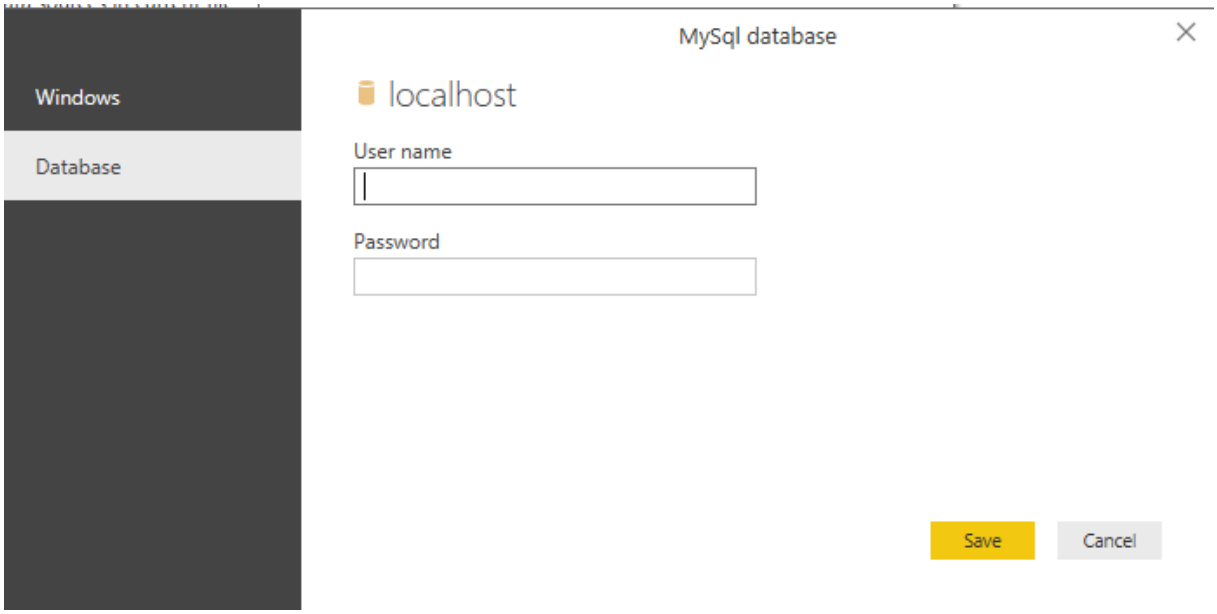
Figur 17.16 Klar for oppkobling til MySql

Gå inn på *Edit Permissions*.



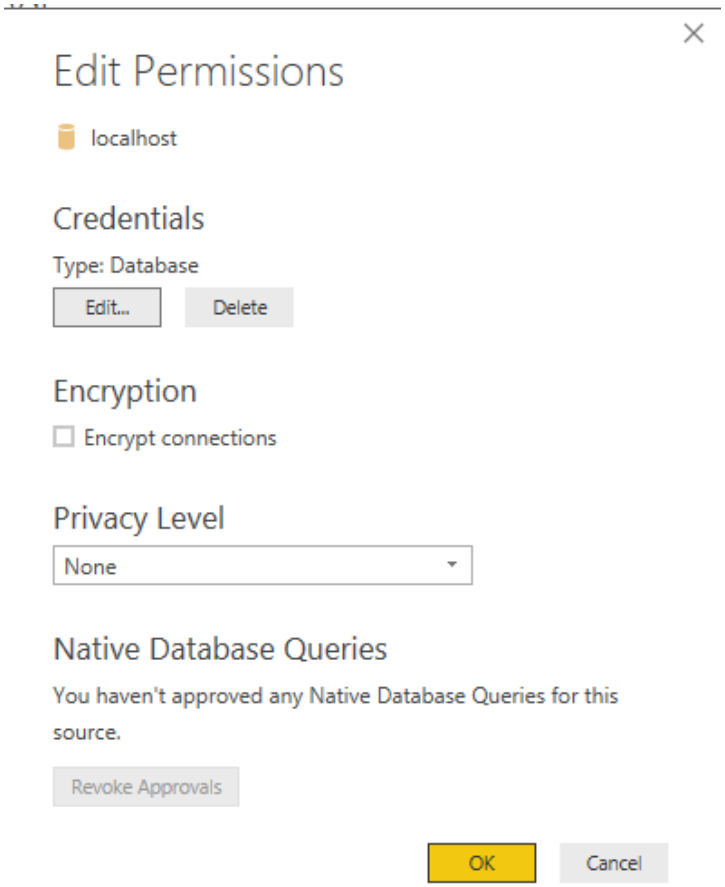
Figur 17.17 Klar for konfigurering

Gå nå inn på Edit under *Credentials*:



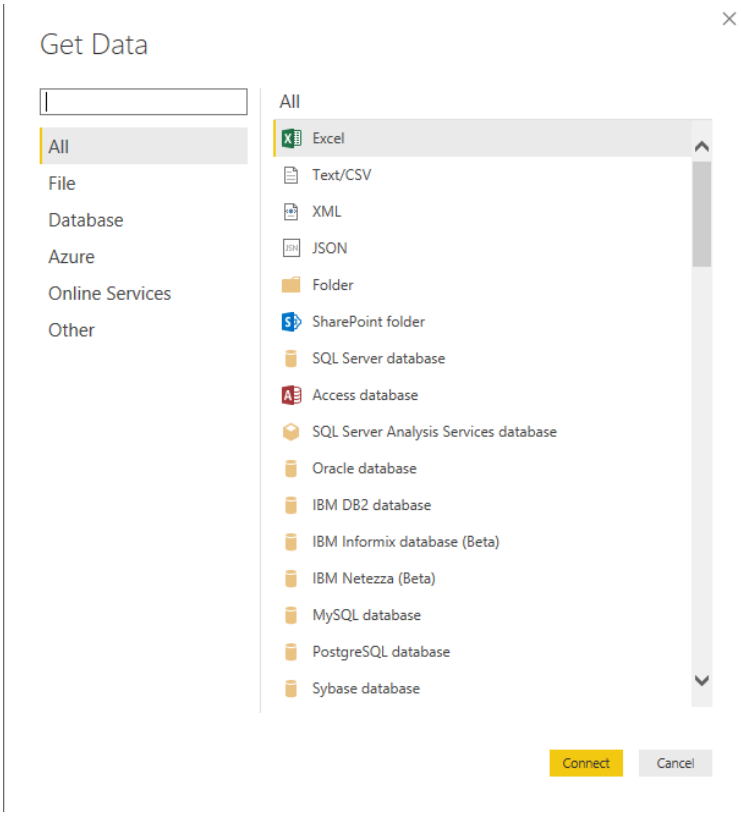
Figur 17.18 Klar for konfigurering av databasekobling

Pass på å markere for Database til venstre i vinduet, og skriv så inn brukernavn og passord. Når du nå klikker på Svae, vil du se at *Credentials* er satt til Database:



Figur 17.19 Neste steg

Nå er vi klar til å importere data fra databasen. Vi velger *Get Data* i Power BI, og velger MySQL database:

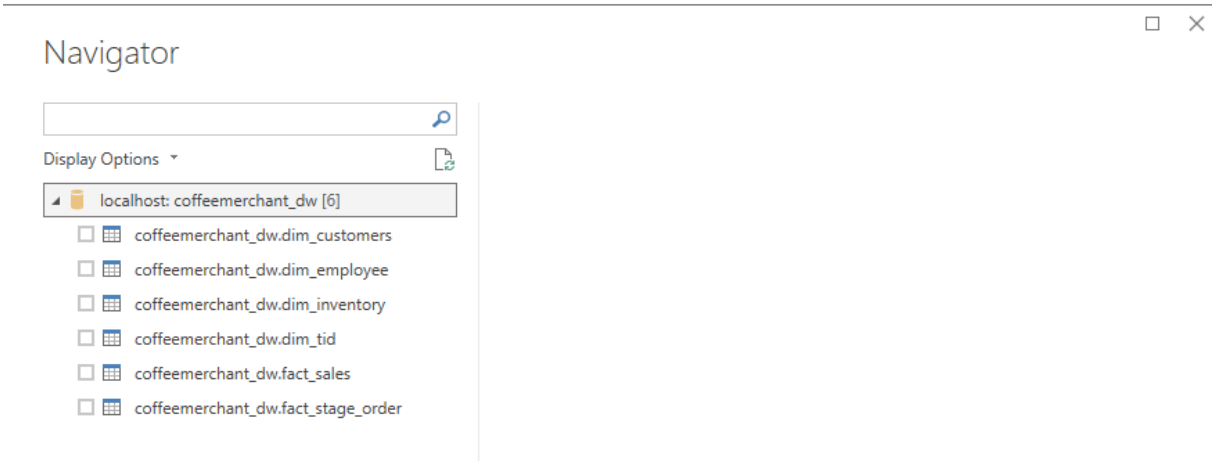


Figur 17.20 Valg av databasesystem

Deretter angir vi server og database:

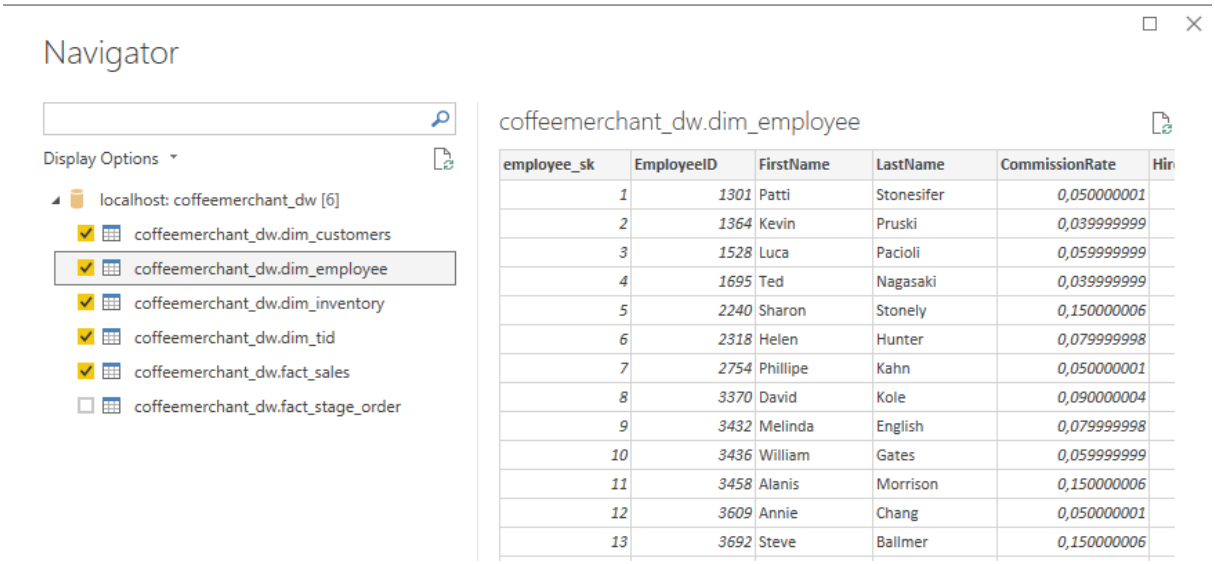
Figur 17.21 Database er valgt

Vi kan nå se alle tabellene i databasen:



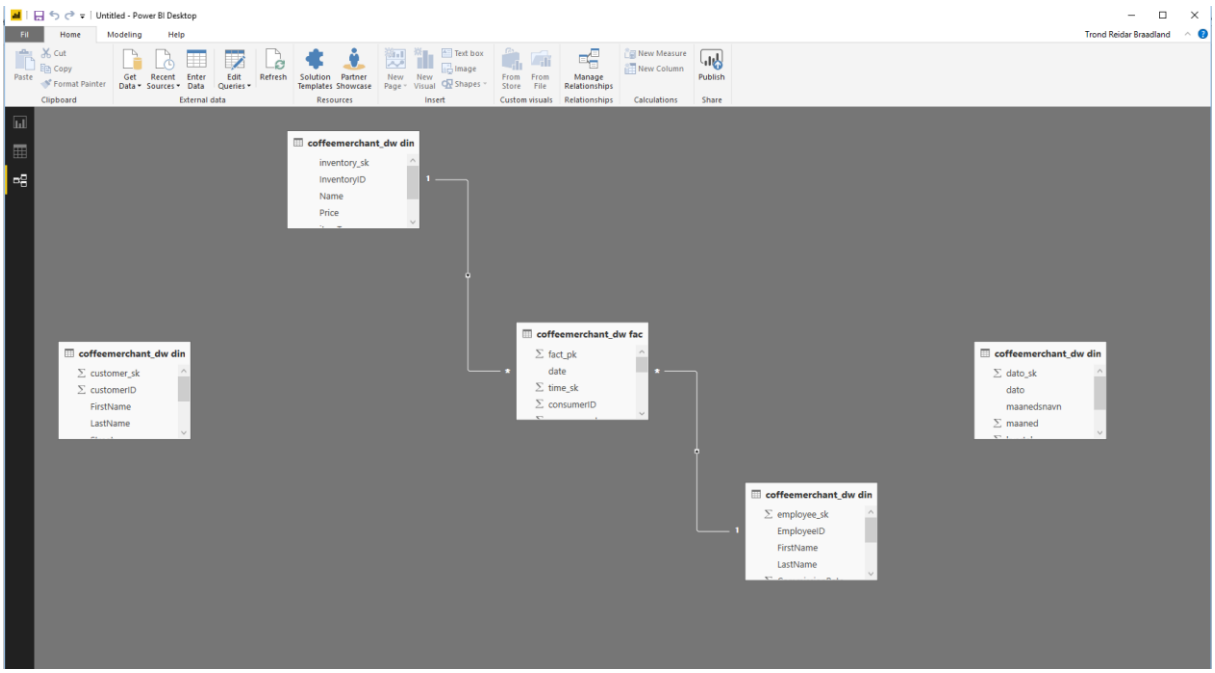
Figur 17.22 Tabellene i databasen vises

Vi markerer tabellene vi skal importere:



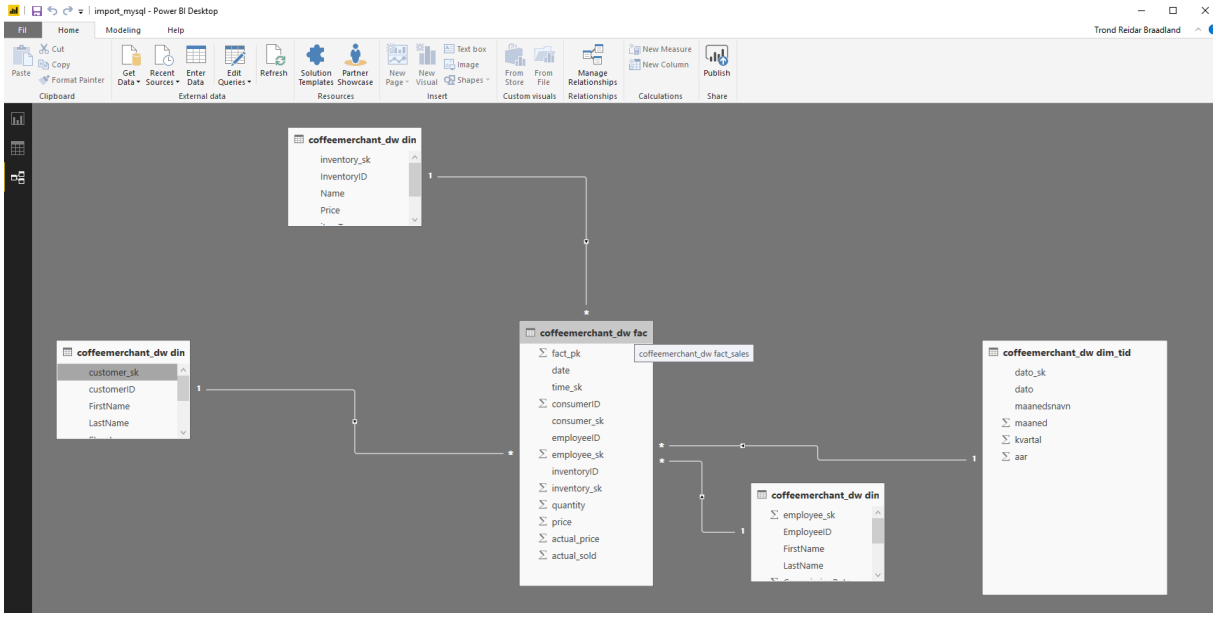
Figur 17.23 Valg av tabeller for import

I Relations-vinduet kan vi nå se databasen og relasjonene:



Figur 17.24 Initiell datamodell vises

Power Bi «forstår» at nøkler med samme navn hører sammen. I to av tabellene er det imidlertid forskjellige navn på primærnøkkel og fremmednøkkel. BI kan opprette relasjonene manuelt ved å klikke på fremmednøkkelen og dra over til primærnøkkelen (eller omvendt):



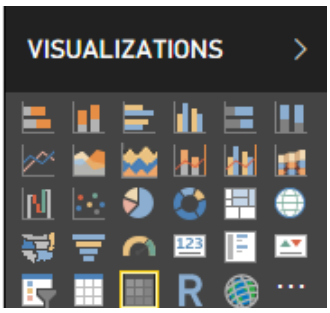
Figur 17.25 Ferdig datamodell

Nå er stjerneskjemaet opprettet i Power BI.

Som tidligere bør vi sjekke datatypene for attributter som har summetegn foran seg. Bare fakta skal være summerbare.

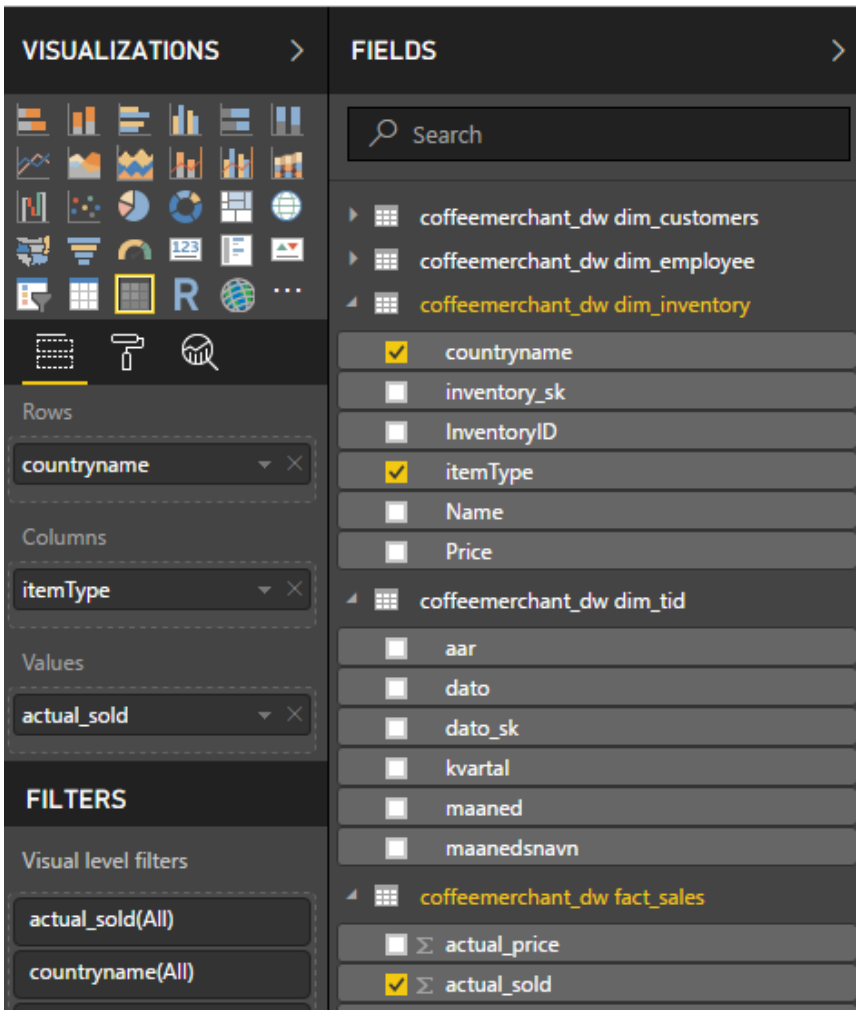
17.4 Analyser basert på stjerneskjema

Det er nå vi virkelig kan se styrken i Power BI. La oss si at vi skal lage en krysstabell med salg av varegrupper fordelt på varens opprinnelsesland. Land og varegruppe finner vi i tabellen dim_inventory, mens salg i dollar fremgår av actual_sold i tabellen fact_sales. Vi velger først at visualiseringen skal være en matrise (symbolet som er rammet inn med gult):



Figur 17.26 Visualiseringer i Power BI

Deretter krysser vi av for det vi skal ha med i datavinduet:



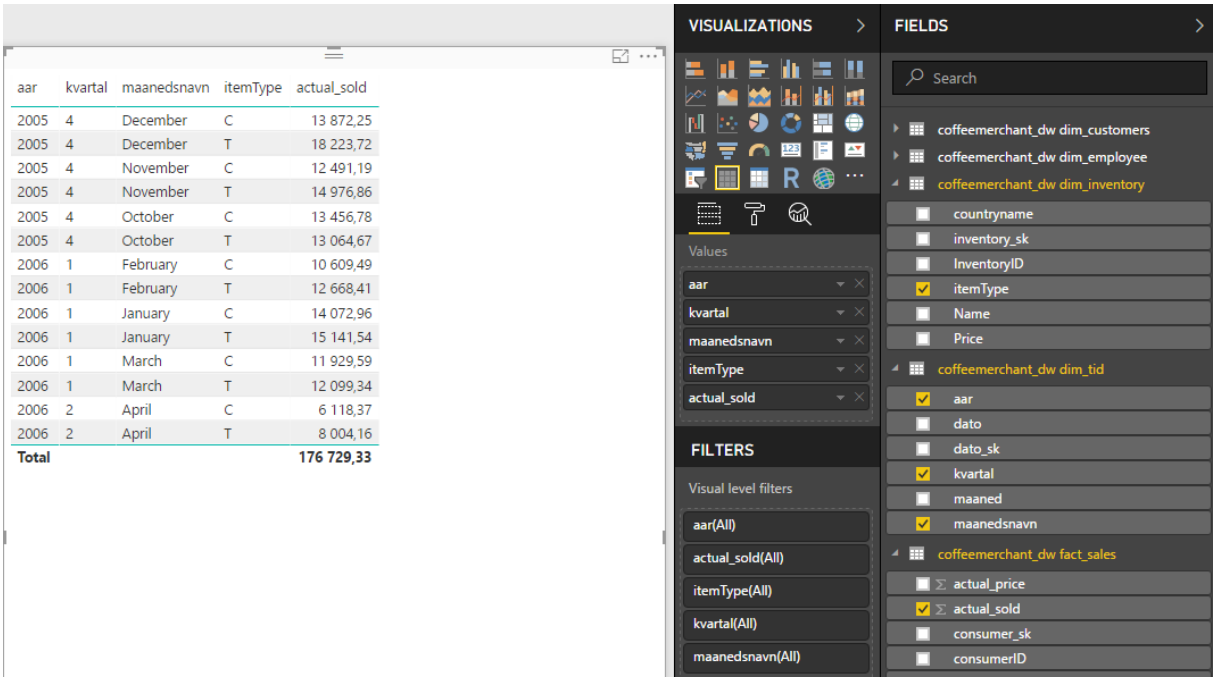
Figur 17.27 Valg av felt i databasen

Her krysset jeg av for countryname først, og Power BI lar automatisk dette være Rows. itemType ble automatisk Columns, og actual_sold Values. Resultatet ser slik ut:

countryname	C	T	**Total**
	39 672,17	20 497,18	**60 169,35**
Brazil	2 812,39		**2 812,39**
China		25 393,75	**25 393,75**
Colombia	2 541,85		**2 541,85**
Costa Rica	2 670,14		**2 670,14**
Egypt		4 424,59	**4 424,59**
Ethiopia	4 382,74		**4 382,74**
Germany		4 169,90	**4 169,90**
Guatemala	1 783,49		**1 783,49**
Haiti	567,20		**567,20**
India		11 989,71	**11 989,71**
Indonesia, Republic of	7 545,78		**7 545,78**
Italy	595,93		**595,93**
Jamaica	2 355,10		**2 355,10**
Japan		13 379,10	**13 379,10**
Kenya	1 935,02	6 518,19	**8 453,21**
Mexico	1 218,37	1 395,73	**2 614,10**
Morocco		1 038,77	**1 038,77**
New Zealand		1 309,84	**1 309,84**
Nicaragua	2 274,68		**2 274,68**
Papua New Guinea	1 206,94		**1 206,94**
Peru	1 478,34		**1 478,34**
Russia		1 989,38	**1 989,38**
Saudi Arabia	2 860,39		**2 860,39**
Sri Lanka		2 072,56	**2 072,56**
Tanzania, United Republic of	1 677,68		**1 677,68**
Turkey	686,84		**686,84**
United States	2 127,60		**2 127,60**
Yemen	826,56		**826,56**
Zimbabwe	1 331,42		**1 331,42**
Total	**82 550,63**	**94 178,70**	**176 729,33**

Figur 17.28 Tabell laget av Power BI

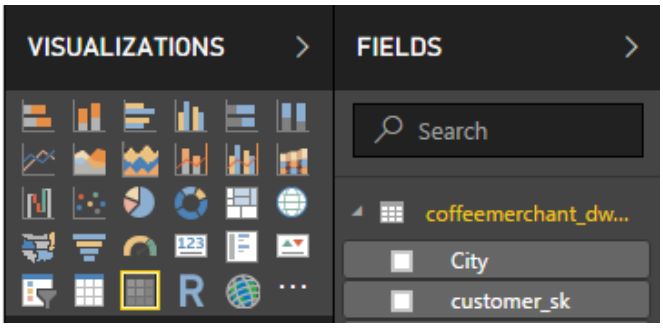
Her har jeg laget er tabell med år, kvartal og månedsnavn, for deretter å vise salg fordelt på varegruppene:



Figur 17.29 Ny tabell

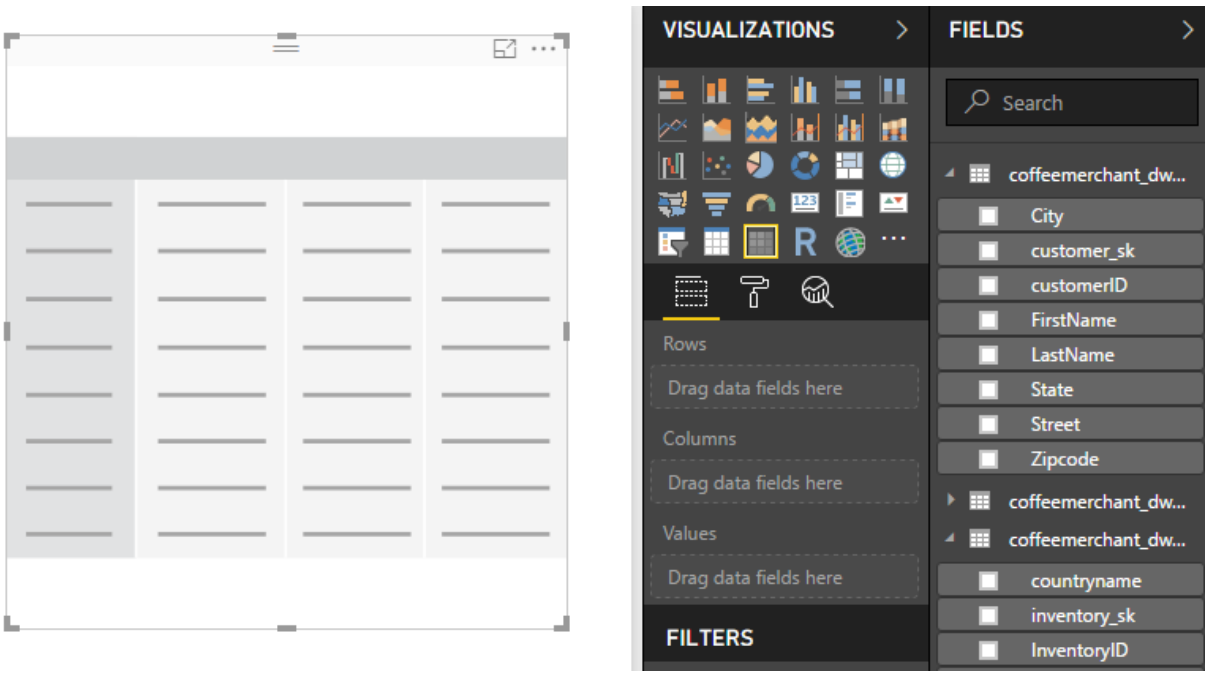
17.5 Drilling med Power BI

Vi skal lage en visualisering med drilling. Vi skal drille på hierarkiet State -> City, og ha fordeling av salg fordelt på varegrupper. Vi velger da en matrise som visualisering, her markert med gul ramme:



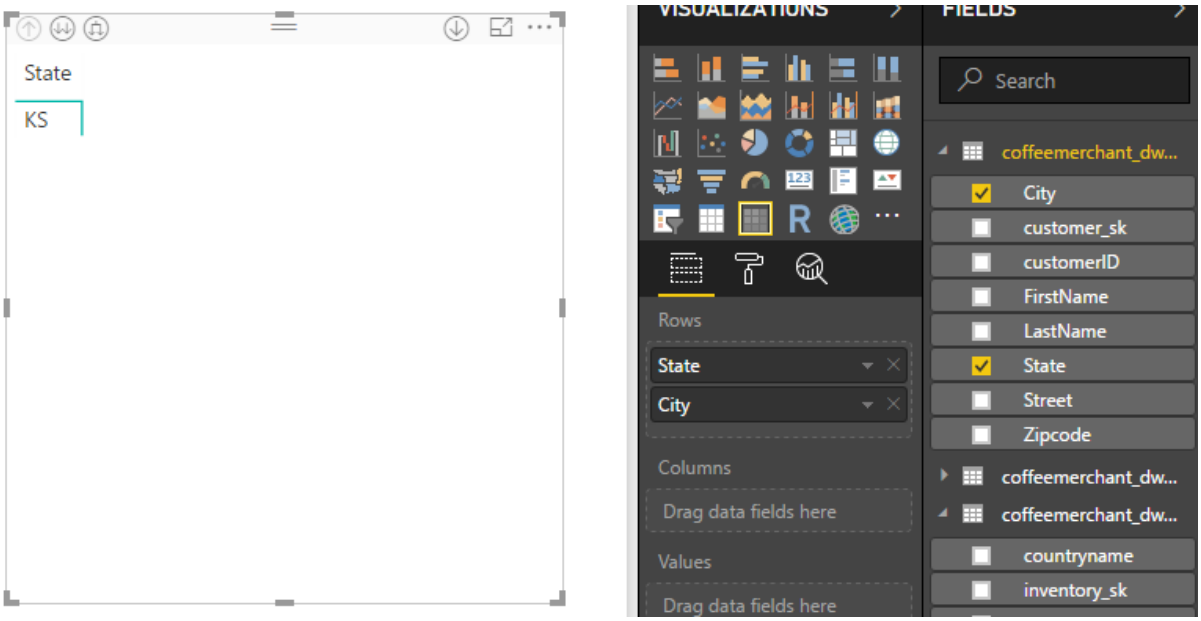
Figur 17.30 Valg av matrise

I utgangspunktet ser det slik ut:



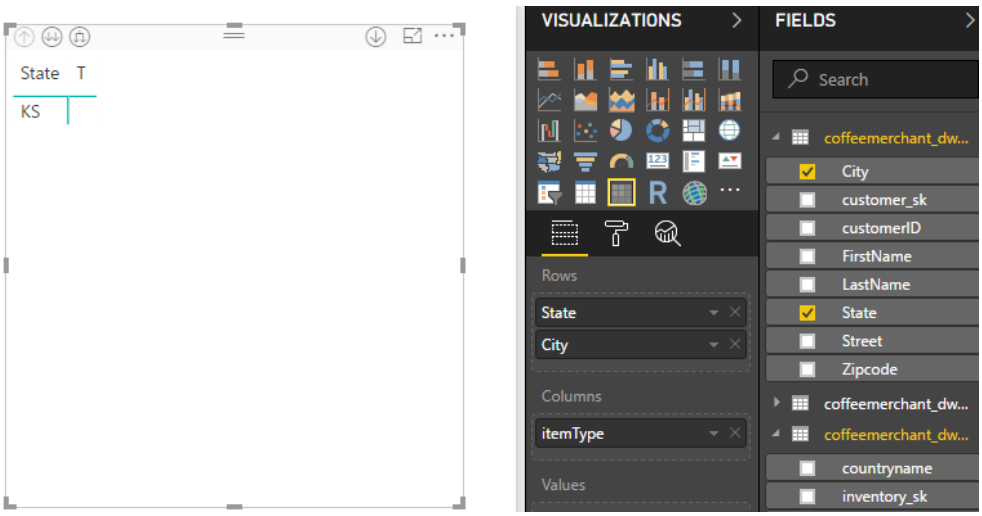
Figur 17.31 Ny matrise

Vi drar så feltene State og City til radfeltet ved siden av tabellisten:



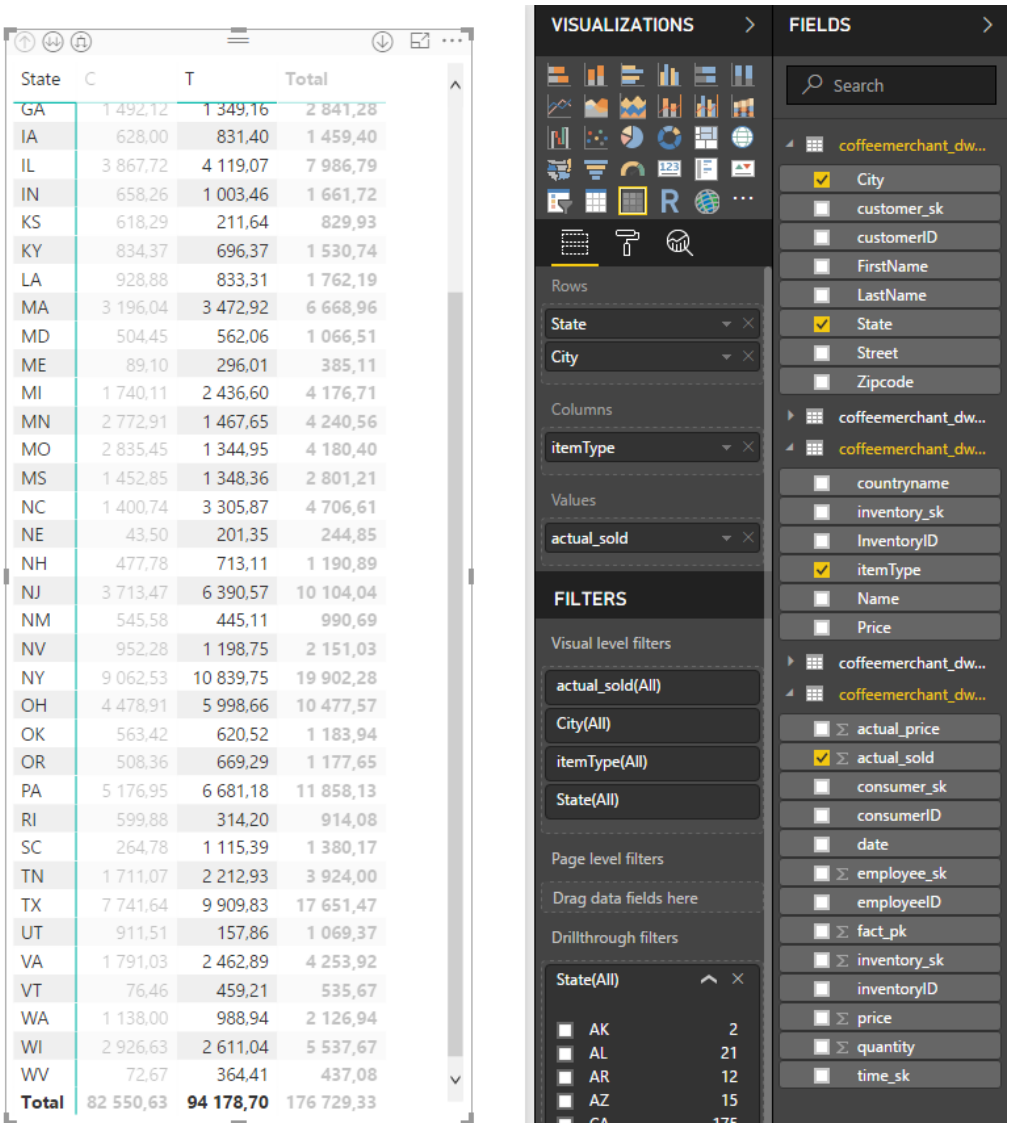
Figur 17.32 Rader er valgt

Deretter drar vi ItemType til kolonnefeltet:



Figur 17.33 Kolonner er valgt

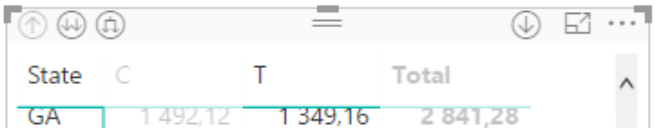
Til slutt drar vi ∑actual_sold til values-feltet og State og City til feltet Drilltrough filters. Resultatet vises nå som ferdig matrise:



Figur 17.34 Summering er valgt

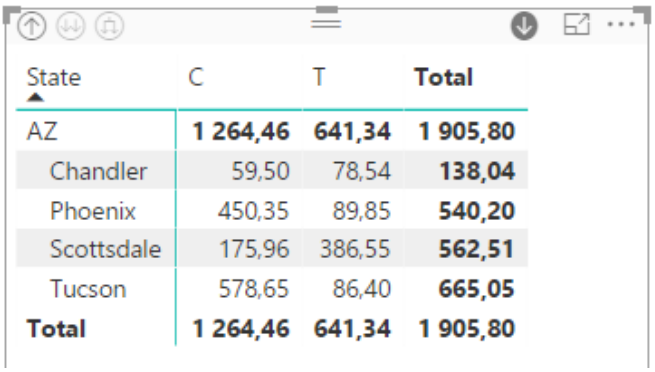
Power BI har også lagt alle de valgte attributtene inn som filtere.

Legg merke til symbolene øverst i matrisen:



Figur 17.35 Symboler for drilling

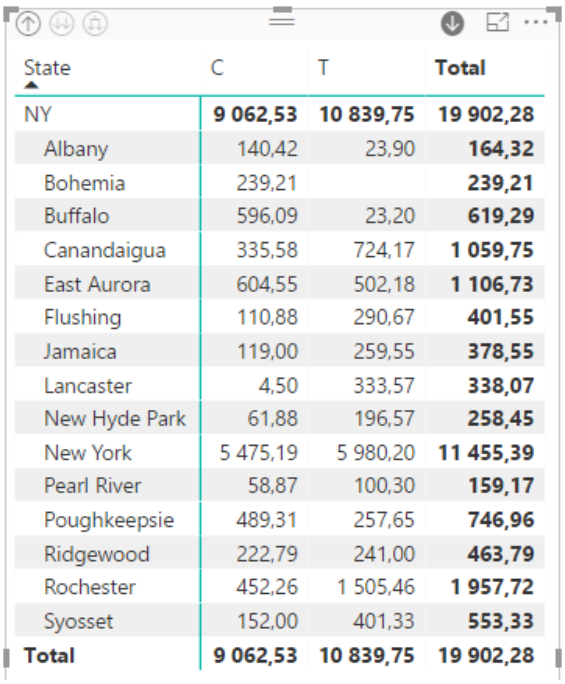
Til høyre ser vi en nedoverpil inne i en sirkel. Dette er symbolet for drill down. Til vensttre ser vi en oppoverpil inne i en sirkel. Dette er symbolet for roll up. Vi kan nå foreta en drill down på matrisen. Vi markerer staten Arizona (AZ), og klikker på Drill down. Resultatet blir dette:



State	C	T	Total
AZ	**1 264,46**	**641,34**	**1 905,80**
Chandler	59,50	78,54	**138,04**
Phoenix	450,35	89,85	**540,20**
Scottsdale	175,96	386,55	**562,51**
Tucson	578,65	86,40	**665,05**
Total	**1 264,46**	**641,34**	**1 905,80**

Figur 17.36 Drilling på valgt stat

Tar vi så en rollup, er vi tilbake til den samlede matrisen, med bare stater. Vi prøver igjen med å drille på staten New York (NY):



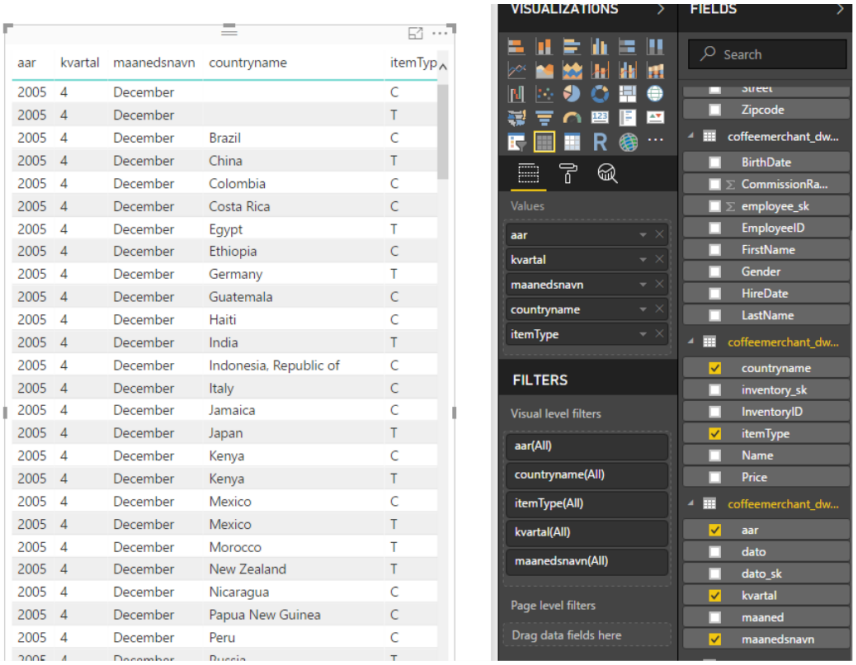
State	C	T	Total
NY	**9 062,53**	**10 839,75**	**19 902,28**
Albany	140,42	23,90	**164,32**
Bohemia	239,21		**239,21**
Buffalo	596,09	23,20	**619,29**
Canandaigua	335,58	724,17	**1 059,75**
East Aurora	604,55	502,18	**1 106,73**
Flushing	110,88	290,67	**401,55**
Jamaica	119,00	259,55	**378,55**
Lancaster	4,50	333,57	**338,07**
New Hyde Park	61,88	196,57	**258,45**
New York	5 475,19	5 980,20	**11 455,39**
Pearl River	58,87	100,30	**159,17**
Poughkeepsie	489,31	257,65	**746,96**
Ridgewood	222,79	241,00	**463,79**
Rochester	452,26	1 505,46	**1 957,72**
Syosset	152,00	401,33	**553,33**
Total	**9 062,53**	**10 839,75**	**19 902,28**

Figur 17.37 Drilling på ny stat

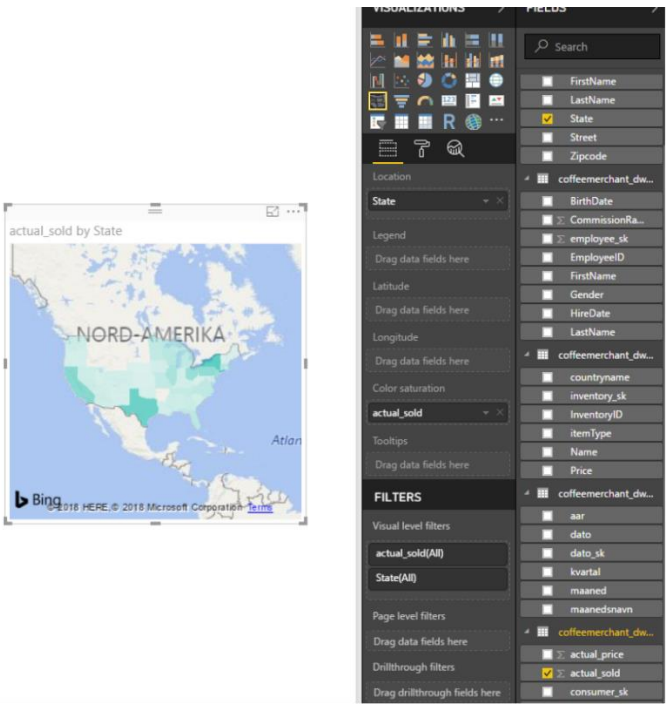
17.6 Dashbord med Power BI

Det kan bli mange rapporter knyttet til et datasett. For å få en enkel tilgang til disse, er det vanlig å lage et dashbord. Dette fungerer som en portal til de enkelte rapportene. Med Power BI kan vi lage de enkelte rapportene med Power BI Desktop, og så publiserer vi dem på Power BI Server. Selve dashbordene lager vi med sistnevnte.

I Power BI Desktop lager vi rapporter som disse:

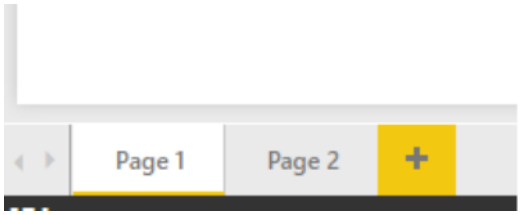


Figur 17.38 Rapport fra Power BI Desktop



Figur 17.39 Kartbasert rapport laget med Power BI Desktop

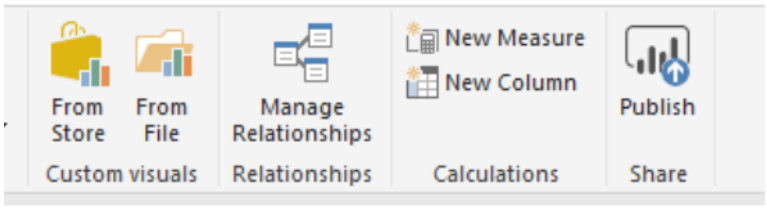
Vi kan lage flere sider med hver sine rapporter. Nederst til venstre på arbeidsbordet ser vi hvordan vi kan legge til nye sider:



Figur 17.40 Legge til ny side

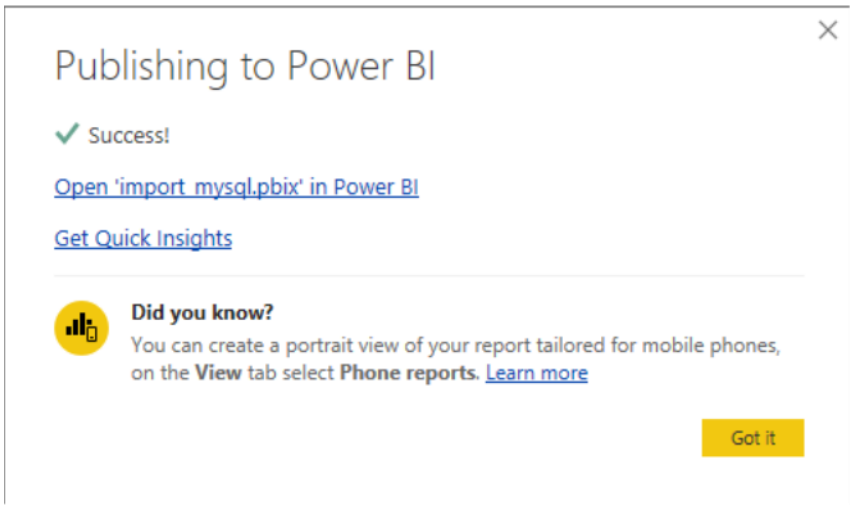
Alle sidene lagres i en felles fil.

Vi kan nå eksportere rapportene våre til Power BI Server. Helt til høyre i menylinjen finner vi knappen **Publish**:



Figur 17.41 Publisering av rapport til Power BI Server

Vi må så identifisere oss med Microsoft brukernavn og passord. Når eksporten er ferdig, kan vi gå direkte fra Power BI Desktop til Power BI Server:



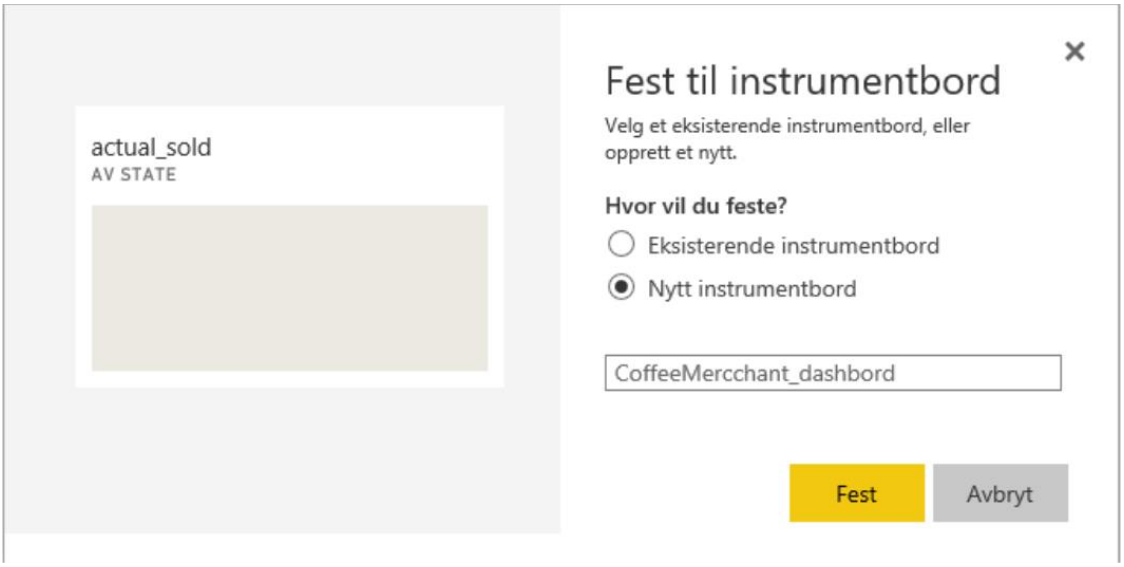
Figur 17.42 Etter publisering

Klikk på Open import mysql.pbix in Power BI (mysql.pbix er navnet jeg har gitt denne filen):

Vi kan nå feste en rapport til et dashbord ved å klikke på pin-symbolet øverst til høyre i rapporten:

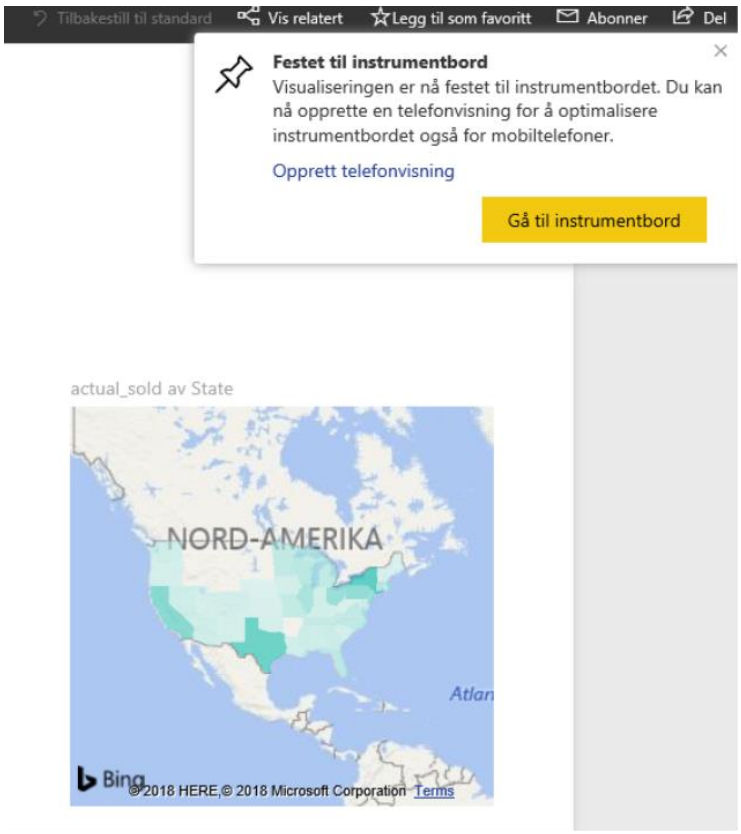
Figur 17.43 Legge til rapport til dashboard i Power BI Server

Vi blir så spurt om hvilket dashbord vi skal legge til rapporten i. Her har jeg valgt å opprette et nytt dashbord:



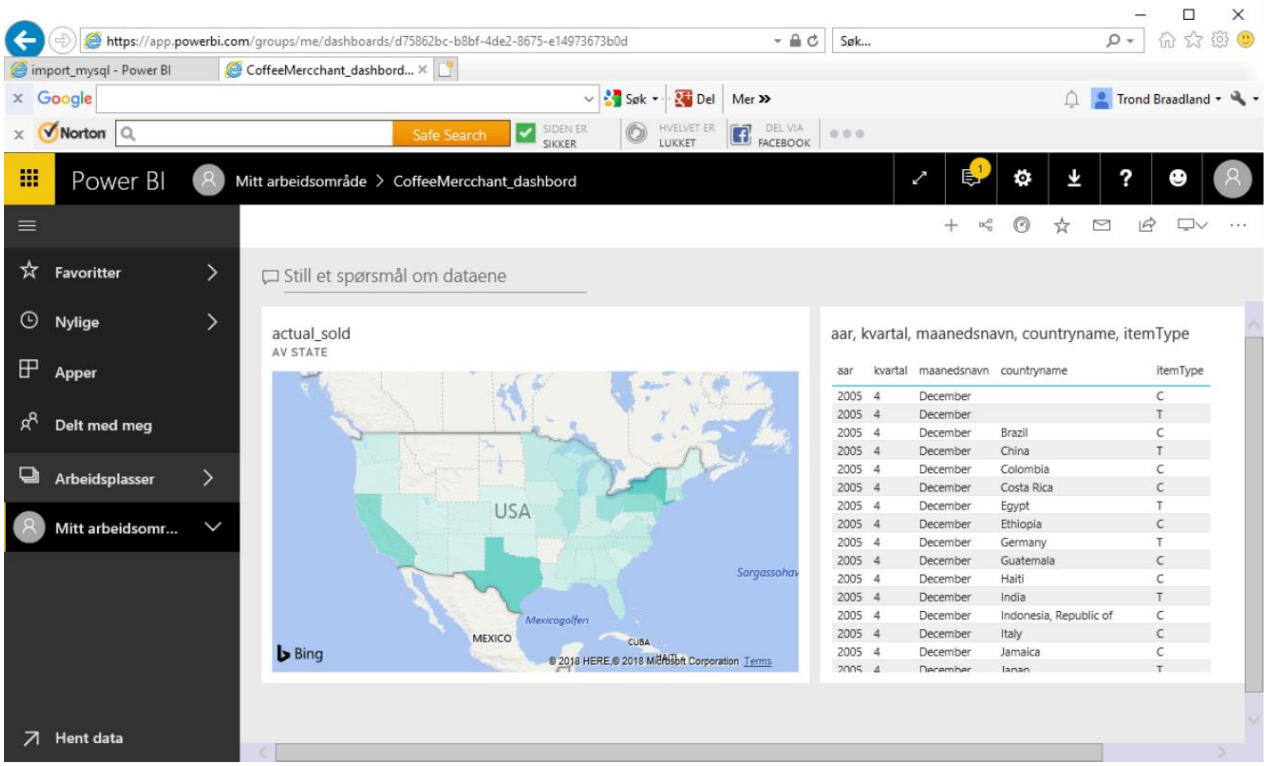
Figur 17.44 Valg av dashboard

Når vi har lagt til de rapportene vi ønsker kan vi gå til dashbordet:



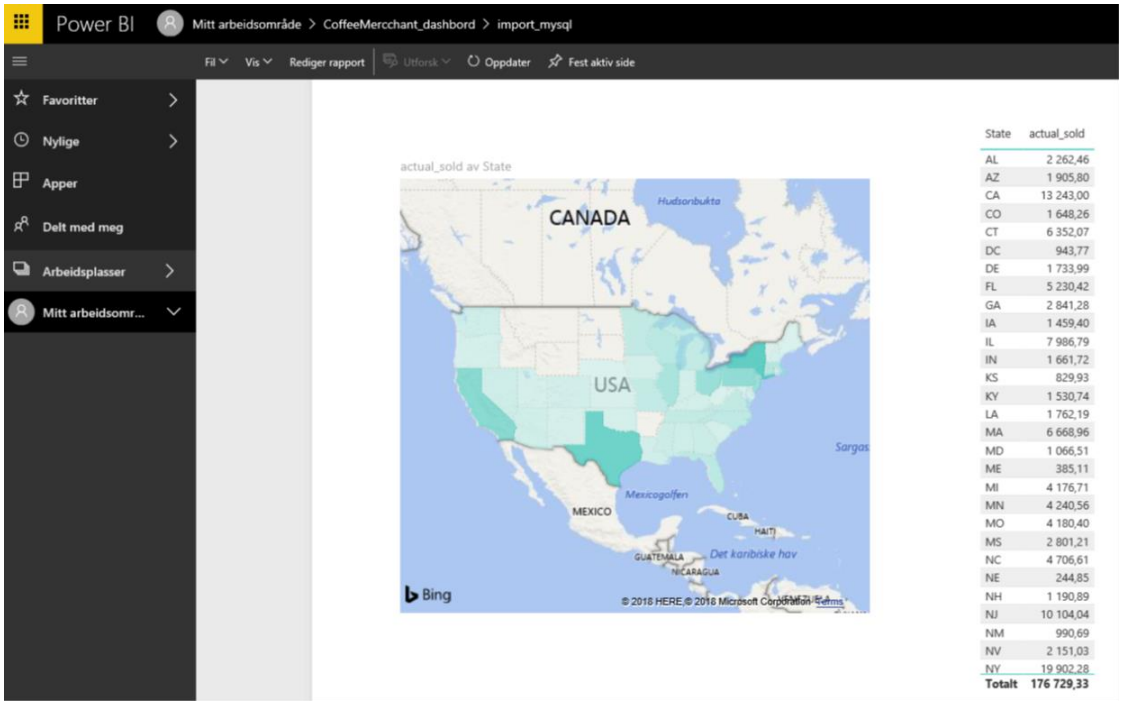
Figur 17.45 Gå til dashboard

Dashbordet ser nå slik ut, etter at to rapporter er lagt til:



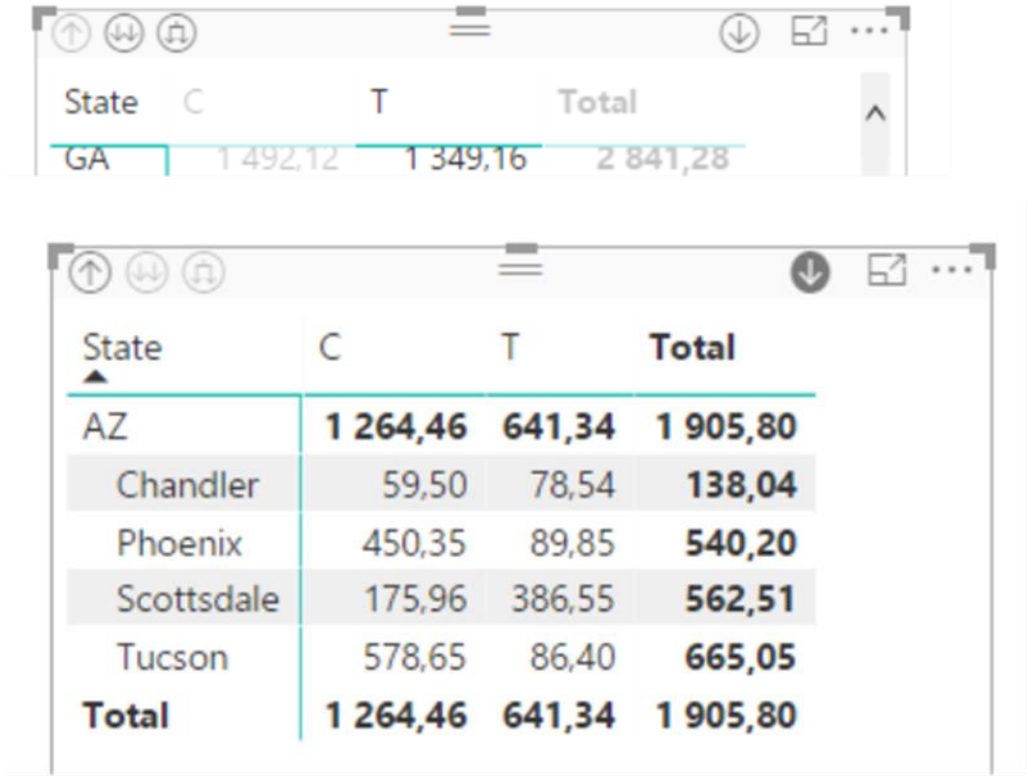
Figur 17.46 Dashbordet

Klikker vi på et av feltene i dashbordet (kalt tiles på engelsk), sendes vi til den aktuelle siden. Merk at på denne siden er det lagt til en rapport det ikke er lenket til fra dashbordet:



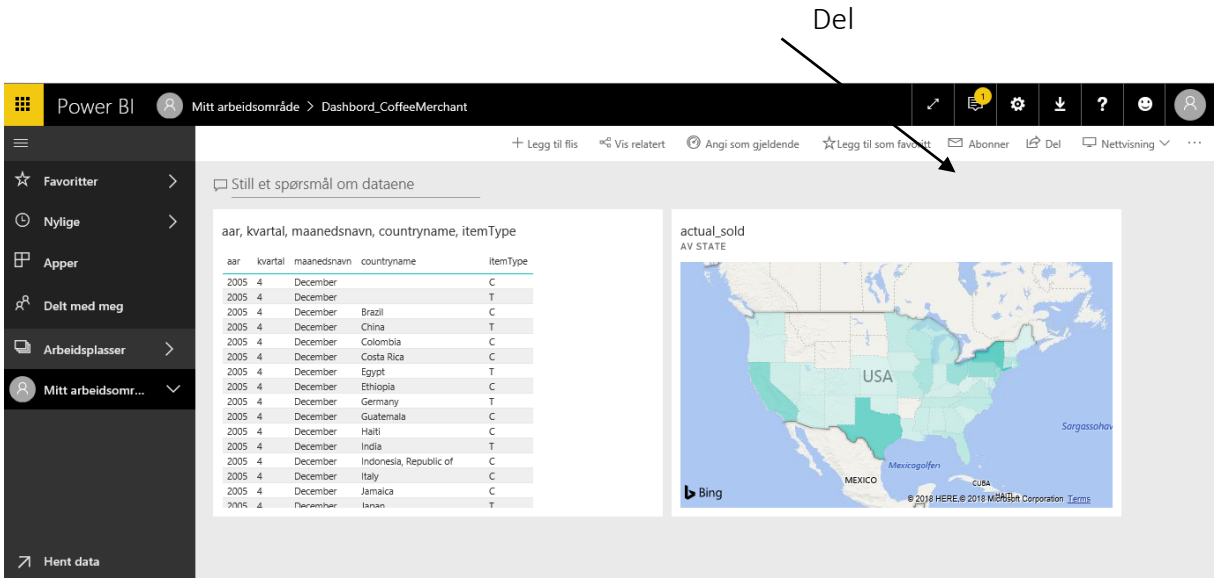
Figur 17.47 Stater i dashboardet

Merk at hvis vi har en rapport med drilling, er drillingen fortsatt tilgjengelig fra nettsiden på Power BI server:



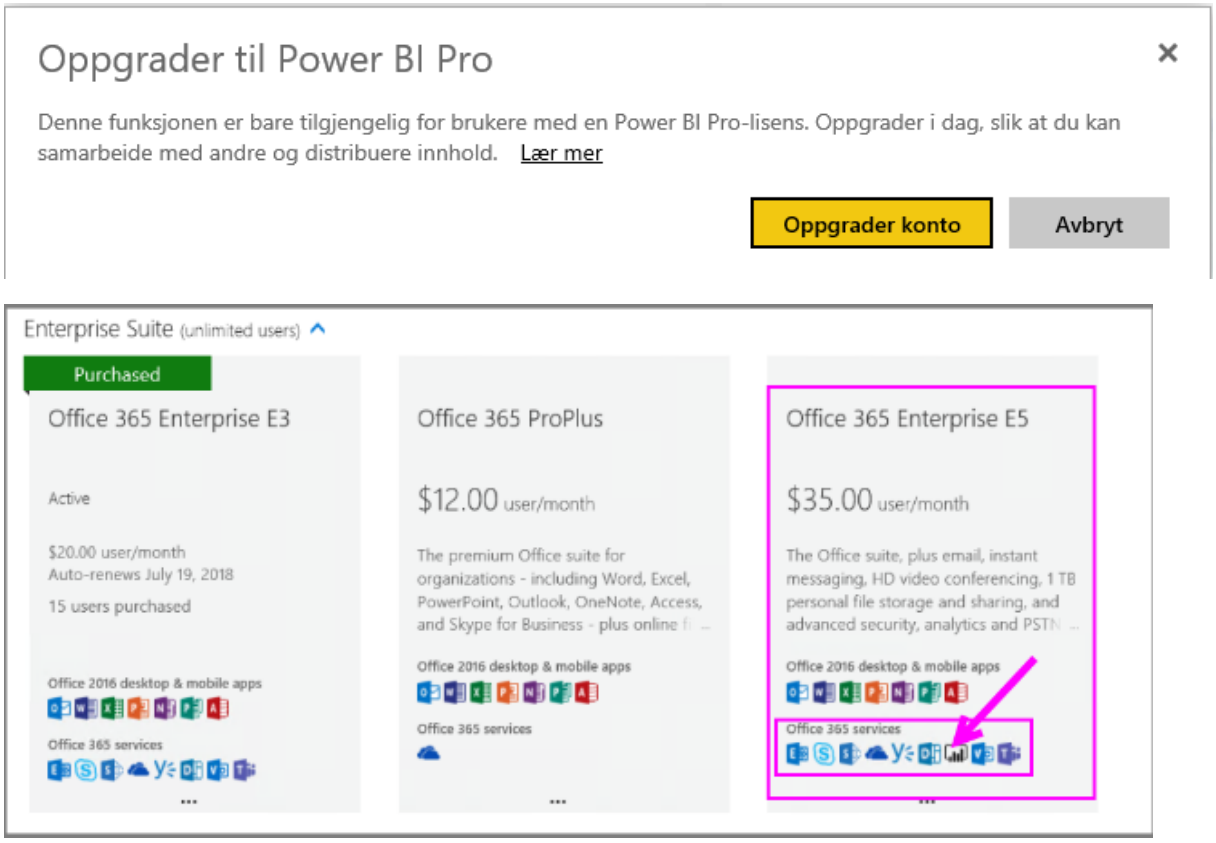
Figur 17.48 Drilling på Power BI Server

Neste punkt er å dele dashbordet med andre brukere:



Figur 17.49 Deling med andre brukere

Men akk, det går ikke. Da må vi ha en betalt lisens:



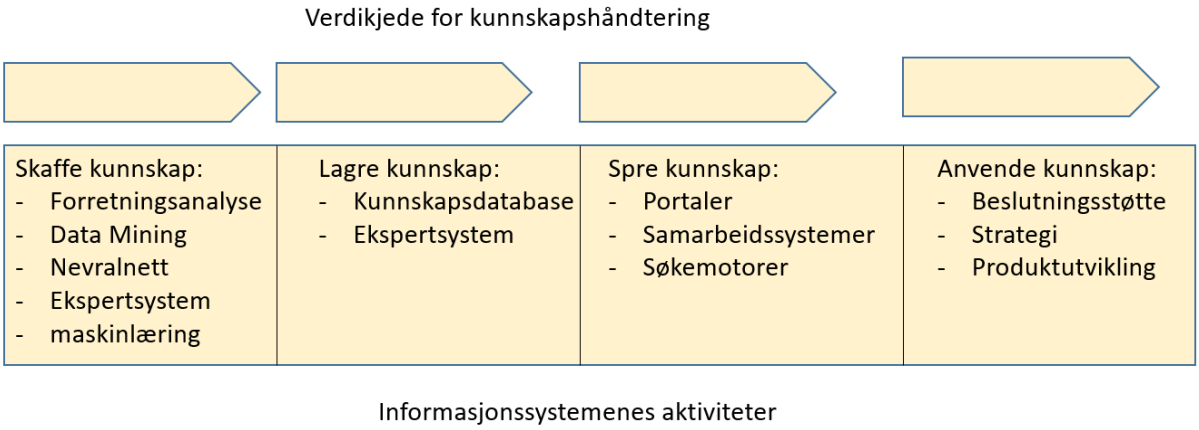
Figur 17.50 Kjøp av lisens på Power Bi

Kapittel 18 Kunnskapsarbeid og kunstig intelligens

Mye av arbeidet i moderne virksomheter er knyttet til kunnskap og data. I offentlig forvaltning er arbeid med kunnskap det sentrale, men også i mange bedrifter er kunnskapsarbeid så sentralt at vi snakker om kunnskapsbedrifter. I slike virksomheter er forvaltning av kunnskap en så viktig ledelsesoppgave at vi snakker om kunnskapsledelse.

18.1 Kunnskapsarbeid og informasjonsteknologi

Som vi har sett, er bruk av informasjonsteknologi sentralt i lagring av informasjon. Det er imidlertid ikke bare i lagringen teknologien spiller en rolle. Vi kan si at informasjonsteknologiens oppgave er å samle inn, kode, behandle, lagre, presentere og distribuere informasjon. Alle disse oppgavene inngår i teknologiens støtte til kunnskapsledelse.



Figur 18.1 Verdikjede for kunnskapshåndtering (etter Laudon 2017)

Laudon og Laudon (Laudon 2017) har presentert ovenstående modell for bruk av informasjonsteknologi i kunnskapsarbeid. I modellen skiller man mellom fire forskjellige typer bruk av IT: til å skaffe kunnskap, til å spre kunnskap, til å lagre kunnskap og til å anvende kunnskap. Til hvert bruksområde hører bestemte typer informasjons-systemer:

- Skaffe kunnskap: programvare for analyse (som OLAP), Data Mining, maskinlæring og koding for bruk i ekspertsystemer skaffer virksomheten kunnskap.
- Lagre kunnskap: kunnskapen må lagres slik at man kan finne den igjen. Egne databaser for dette kan brukes (for eksempel Hadoop-filer). I et ekspertsystem lagres kodet kunnskap for bruk i ressonering.
- Spre kunnskap: nettbaserte portaler, typisk med dashboard, er gjerne inngangen for brukerne. Mange systemer tilbyr teknologi for samarbeid, for eksempel Microsoft Outlook og GoogleDocs. Tilgang til søkemotorer for søk i kunnskapsdatabasene hører også med.
- Anvende kunnskap: endelig skal kunnskapen brukes til beslutninger på alle plan, til strategiarbeid, produktutvikling med mer.

Vi skal se nærmere på noen av disse typene systemer.

18.2 Kunnskapsarbeidssystemer

Kunnskapsarbeidssystemer, på engelsk kalt Knowledge Work Systems (KWS) er systemer som hjelper kunnskapsarbeidere, dvs de som skal skape ny kunnskap. Kunnskapsarbeidssystemer spenner over et vidt spekter, fra dataassistert konstruksjon (DAK) til statistikkprogrammer som SPSS. Typisk er at dette er applikasjoner som kjøpes ferdig. I noen tilfelle, som med avanserte DAK-systemer, hører det med egen hardware.

De mest utbredte systemer i denne kategorien er nok DAK-systemer. Disse gjør det mulig å konstrukere maskiner, hus og annet ved en datamaskin i stedet for ved et tegnebord. Systemene finnes i en rekke utgaver fra forskjellige leverandører. Noen kan kjøres på en vanlig PC, som AutoCad. Andre krever spesielle arbeidsstasjoner med to skjermer og en server for lagring av konstruksjonsdata. Det finnes også ferdige bransjeløsninger, for eksempel for konstruksjon av store prosessanlegg.

DAK kombineres i dag ofte med data fra geografiske informasjonssystemer (GIS). Ved konstruksjonen av OL-anleggene på Lillehammer brukte man for eksempel GIS-data for å konstruere anleggene i terrenget.

Intergraph leverer systemer spesielt egnet for konstruksjon av store industrianlegg. De kaller sitt system PDS – for Plant Design System

Et avansert DAK-system kan ikke bare skrive ut konstruksjonstegninger. En annen viktig funksjon er at det kan presentere tredimensjonale bilder av det man har konstruert. Disse brukes som såkalte walk-troughs, dvs at man kan simulere en vandring gjennom konstruksjonen. På denne måten kan man oppdage feil og mangler. Figur 18.5 illustrerer en slik fremstilling.

Minst like viktig er den muligheten avanserte systemer har til å utføre kollisjonstester. Ved store prosjekter er det et stort antall mennesker involvert i konstruksjonsarbeidet. Ved konstruksjon av en oljeplattform eller en fabrikk vil typisk forskjellige ingeniørgrupper med ansvar for hvert sitt område, som elektrisk anlegg, prosessanlegg, bygningskonstruksjon eller røranlegg være involvert. Konstruksjonen er så stor at den enkelte ingeniør ikke kan ha oversikt over hva de andre til enhver tid har gjort. Resultatet er at der noen har tegnet inn en trapp kan en annen ha tegnet inn et halvmetertykt rør. Dette kalles kollisjon, og før DAK-systemer ble tatt i bruk måtte man ha en manuell gjennomgang av konstruksjonstegningene for å oppdage slike. Mange av kollisjonene ble allikevel ikke oppdaget før under byggingen, da man plutselig oppdaget at det allerede stod en trapp der tegningen sa at det skulle være et rør. Avanserte DAK-systemer oppdager selv slike kollisjoner, noe som effektiviserer arbeidet svært mye. Siden vi her snakker om prosjekter der bare konstruksjonsarbeidet kan koste en milliard kroner, er det snakk om store effektiviseringsgevinster.

Forskere bruker flere typer programvare, spesielt statistikkprogrammer som SPSS. Andre typer programvare kan for eksempel være Mathematica. Dette er applikasjoner som brukes til statistiske og matematiske analyser, noe forskere typisk må gjøre.

Andre typer kunnskapsarbeidssystemer kan være applikasjoner som hjelper komponister med å skrive musikk, eller advokater med å utarbeide saksinnlegg. I det hele tatt er det i dag applikasjoner som støtter de fleste typer kunnskapsarbeid (vi regner allikevel ikke Word som et kunnskapsarbeidssystem, selv om forfattere bruker det).

18.3 Kunstig intelligens

Kunstig intelligens er et av de temaer det er blitt fokusert mye på når det gjelder informasjonsteknologi. Drømmen – eller mareritttet – om selvstendig tenkende datamaskiner oppstod tidlig. Temaet er blitt flittig brukt i science fiction, men også eksperter innen informatikk har hatt dristige vyer. I virkeligheten har det vist seg å være svært komplisert å gi en datamaskin intelligens, delvis fordi vi faktisk ikke vet ordentlig hvordan vår egen hjerne virker. Det har imidlertid kommet en del brukbare informasjonssystemer ut av forskningen på kunstig intelligens. Det er dette vi skal se nærmere på i dette kapittelet.

Intelligens hos en datamaskin kan oppnås enten ved at selve maskinen bygges på en spesiell måte, eller ved at man bruker spesiell programvare, eller som en kombinasjon. I dette kapittelet er derfor uttrykkene datamaskin, informasjonssystem og programvare brukt om hverandre.

18.3.1 Hva er intelligens?

De grunnleggende spørsmålene er selvsagt hva vi mener med intelligens, og hvordan den oppstår. Det finnes flere definisjoner på intelligens, og de fleste er knyttet til at det handler om å kunne løse *nye* problemer og å lære av erfaringer. Det å kunne løse problemer etter en gitt oppskrift er altså ikke intelligens, heller ikke hvis man ikke lærer av de erfaringene man gjør.

Det finnes også en teori som sier at menneskelig intelligens består av syv forskjellige funksjoner: språklig intelligens, musikalsk intelligens, matematisk-logisk intelligens, romlig intelligens, kroppslig-bevegelsesmessig intelligens, interpersonlig intelligens og intrapersonlig intelligens. Andre har utvidet modellen med to til tre typer til. Vi skal ikke her gå nærmere inn på hva disse forskjellige intelligensfunksjonene går ut på, men bare konstatere at vår forståelse av begrepet både er innviklet og langt fra entydig.

Intelligensbegrepet omfatter ikke selve bevisstheten, det vil si vår evne til å oppfatte oss selv som en egen enhet. Om vi kan se bevissthet og intelligens løsrevet fra hverandre, er også et diskusjonstema. Bevisstheten er noe mennesker har filosofert over siden de gamle grekeres dager. Bevissthetsfilosofi er faktisk en egen gren av faget filosofi.

Helt sentralt for oss som intelligente vesener er vår evne til å bruke språk. Alle verdens språk er like avanserte verktøy for å overføre tanker fra et menneskes hjerne til et annet, enten direkte ved tale eller via et lagringsmedium (skrift). Vi kan ved hjelp av språket ikke bare uttrykke konkrete ønsker (mat! drikke!), men også abstrakte begreper. Ingen dyr har denne evnen.

Evne til å lære er ikke unikt for mennesker. Hos svært mange dyrearter, og da snakker vi om både hvirveldyr og hvirvelløse dyr, finner vi en viss læringsevne. Læring spenner fra enkel stimulus-respons (som Pavlovs hunder, som lærte seg å sikle når de hørte en klokke, fordi de visste at det da ville komme mat) til dannelse av nye begreper og sammenhenger mellom disse. Det er ingen dyr som kommer i nærheten av vår evne på dette området.

Kreativitet er også en viktig side ved vår intelligens. Evnen til å skape noe, og til å finne nye løsninger på problemer, å se ting på en ny måte, er kanskje det som fremfor noe annet karakteriserer menneskelig intelligens.

Både vår intelligens og vår bevissthet er uansett knyttet til hjernen. Dette organet består av rundt 100 milliarder nevroner (hjerneceller) som hver har ca 1000 koblinger til andre hjerneceller [1] [2]. I løpet av vår oppvekst tilegner hjernen seg på egen hånd kunnskap og adferdsmønstre, samtidig som vi kan tilegne oss kunnskap etter eget ønske. Mens man tidligere så hjernen hos et nyfødt barn nærmest som et blankt ark, ser vi nå hjernen som et organ med innebyggede strukturer som er grunnlag for læring. Spesielt vår evne til å lære språk er innebygget.

Vår hjerne skiller seg fra selv de mest avanserte dyr gjennom sin kompleksitet. Det er ikke uten grunn at mange forskere peker på den menneskelige hjerne som den mest kompliserte kjente struktur i universet.

18.3.2 Hva er kunstig intelligens?

En definisjon av kunstig intelligens må ta utgangspunkt i vår forståelse av menneskelig intelligens. Den klassiske definisjonen på kunstig intelligens legger vekt på språkforståelse, evne til problemløsning og evne til læring. Med språkforståelse mener vi at et kunstig intelligens-program kan forstå setninger uttalt i naturlig språk. For noen tiår siden så mange for seg en datamaskin som ikke bare kunne forstå det som ble sagt til den, men som også kunne løse problemer den ikke hadde noen tidligere erfaring med. I tillegg skulle selvsagt maskinen kunne lære. Det var også en utbredt tro på at en slik maskin ville ha en bevissthet.

Forskning har vist at det ikke er så enkelt som man trodde å lage en intelligent maskin. Bare det å få en maskin til å forstå vanlig tale er et enormt problem. Talegjenkjenning, det vil si at en maskin kan læres opp til å kjenne igjen visse lydkombinasjoner som kommandoer, har ikke noe med språkforståelse å gjøre. Vi snakker i stedet om at maskinen kan forstå en setning den aldri har hørt før, uttalt i vanlig språk.

[1] De færreste av oss har noen umiddelbar forståelse av slike tall. Svar fort: hvor lang tid trenger du for å telle til 1 milliard dersom du teller et tall i sekundet? Regn deretter ut hva det blir. Svaret vil gi en viss forståelse for hvor mye en milliard faktisk er

[2] Tallet 100 milliarder er det antallet hjerneceller det hyppigst refereres til. Imidlertid oppgis sjelden noen kilde for dette tallet. Resultatet av forsøk på opptelling av antallet hjerneceller varierer fra ca 85 milliarder til 120 milliarder. I tillegg er det ca 19 ganger så mange glia-celler som utgjør hjernens bindevev.

Problemløsning, i den forstand at vi løser nye problemer, krever også svært mye. Problemløseren må ha evne til å trekke paralleller med tidligere erfarte problemer, til å foreta logiske resonnementer, og forklare både for seg selv og andre hvordan resonnementet var. Prøving og feiling vil ofte inngå i løsningen av nye problemer.

Maskinlæring er et område det forskes mye på. Man kan ved relativt enkel programmering få et program til å lære av "erfaring", men bare innenfor svært avgrensede problemområder. Sjakkprogrammer er et eksempel på dette. Programmeringen kan være enkel statistisk beregning av grad av suksess for forskjellige trekk eller kombinasjoner av disse.

En grunnleggende utfordring er hvordan vi skal representere kunnskap. Hva vil det egentlig si at vi *vet* hvordan vi skal spille sjakk? Det finnes flere måter å gjøre dette på i et dataprogram.

Det store problemet innenfor kunstig intelligens har vist seg å være avgrensning av problemområdet. Ingen kunstig intelligens-program i dag kan brukes til generell problemløsning, og vi har ingen anelse om hvordan vi skal kunne lage noe slikt.

Matematikeren Alan Turing har beskrevet en test for å avgjøre om en maskin er intelligent. Denne testen er kjent som *Turing-testen*, og er basert på en "spørrelek" med tre personer. En person et utspørrer, de to andre er i utgangspunktet en mann og en kvinne. Den ene av disse skal prøve å lure utspørreren til å tro at vedkommende er av motsatt kjønn, mens den andre ikke skal prøve på noe lureri. Utspørreren skal så ved å stille spørsmål avgjøre hvem som er hvem. Selve Turing-testen går ut på å erstatte en av disse to personene med en datamaskin. Hvis maskinen klarer å lure utspørreren like ofte som et menneske, er maskinen intelligent. Det sentrale her er at maskinen kan delta i en konversasjon akkurat slik et menneske ville. Hittil har intet kunstig intelligens-program vært i nærheten av å klare denne testen.

Forskning innen kunstig intelligens har imidlertid ført til utvikling av en rekke nyttige systemer som etterligner menneskelig resonnering. Vi kaller disse systemene *ekspertsystemer*. De kan brukes innenfor klart definerte problemområder, der reglene for problemløsning riktig nok er kompliserte, men allikevel mulige å kartlegge fullt ut. Vi skal se nærmere på hvordan denne typen systemer er bygget opp.

Det finnes også andre typer systemer innenfor kunstig intelligens, som nevrale nett og genetiske algoritmer. Vi skal også se kort på disse.

18.3.3 Ekspertsystemer

Et ekspertsystem er altså et informasjonssystem som er laget for å etterligne menneskelig problemløsning innen et avgrenset, veldefinert problemområde. Eksempler på slike problemområder er feilsøking i datanettverk og medisinsk diagnostisering. Prinsippet er at systemet skal stille spørsmål til brukeren, og ut fra svarene velge nye spørsmål til det kan trekke en konklusjon. Det skal deretter presentere konklusjonen, og forklare hvordan det kom frem til denne.

Det er to grunnleggende forskjellige måter å konstruere ekspertsystemer på: regelbaserte og Frame-baserte. I et regelbasert ekspertsystem legger man inn regler av typen HVIS slik og slik ELLERS så og så. Dette er samme prinsipp som når man programmerer valg i et vanlig program, forskjellen er at ekspertsystemet kan håndtere tusenvis av slike regler. Ved vanlig programmering må antall alternativer i en valgsetning være lavt, ellers blir programkoden fort uhåndterlig. I regelbaserte ekspertsystemer er altså kunnskapen representert som regler.

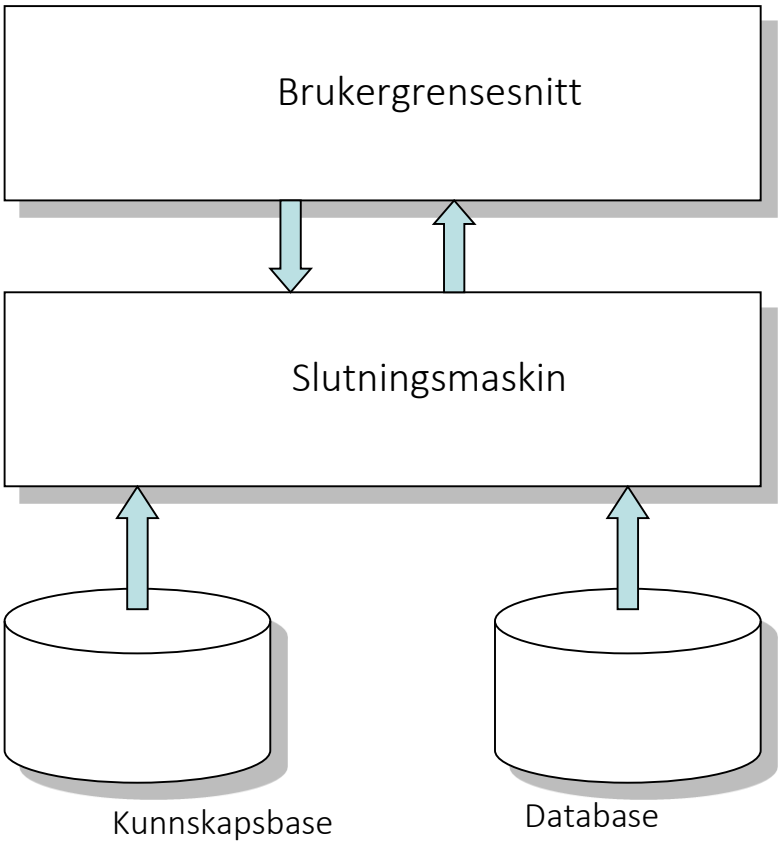
Frames representerer begreper, sammenhenger mellom disse og de egenskaper de har. Regler kan også bygges inn i en Frame. Et eksempel på en Frame kan være et kjøretøy. Det har visse egenskaper, og sammenheng med andre begreper som for eksempel eier og motor. Hvis visse betingelser er opfylt, er kjøretøyet en personbil, hvis andre er oppfylt er det en lastebil. I dette tilfellet er altså kunnskap representert som Frames.

Uansett hvordan kunnskapen er representert, trenger ekspertsystemet en programmodul som kan trekke slutninger på grunnlag av den. En slik modul kalles gjerne en *slutningsmaskin*. Det er denne som sørger for å stille spørsmål og trekker konklusjoner ut fra de svarene den får.

Ekspertsystemer fungerer som erstatning for en menneskelig ekspert. En slik erstatning kan ha mange fordeler. Ofte har man rett og slett ikke tilgang på en menneskelig ekspert. Et ekspertsystem blir heller ikke trett eller uoppmerksom, og egner seg derfor til overvåkningsoppgaver som krever ekspertkunnskap. Et ekspertsystem vil imidlertid ikke kunne erstatte et menneske på permanent basis. Det er ikke i stand til å tilegne seg ny kunnskap, bare følge de regler som er matet inn i det. Det er menneskelige eksperter som bedrar med dette.

Utvikling av et ekspertsystem går først og fremst ut på å kartlegge hvordan en menneskelig ekspert resonnerer ved problemløsning. Denne prosessen kalles *kunnskapsakvisisjon*. Når man vet hvordan mennesket arbeider, legges dette inn i ekspertsystemets kunnskapsbase.

Oppbygningen av et ekspertsystem kan vi beskrive med en figur:



Figur 18.1 Oppbygning av et ekspertsystem (etter Kristensen 1997)

Kunnskapsbasen inneholder regler eller Frames. Databasen vil være en vanlig database med fakta.

En vanlig fremgangsmåte når man skal lage selve systemet er å bruke et såkalt ekspertsystemskall. Dette er programvare som består av slutningsmaskin, grensesnitt og databaseverktøy. I dette skallet legger man så kunnskapsrepresentasjonen.

Man kan også programmere ekspertsystemet fra grunnen av. Det finnes egne programmeringsspråk som er laget for dette formålet, best kjent er Prolog og Lisp. I dag brukes også objektorienterte programmeringsspråk, først og fremst når kunnskapsrepresentasjonen er basert på Frames.

”Kunstig intelligens”, enten i form av ekspertsystemer eller en av de andre teknologiene, er allerede i praktisk bruk. Ekspertsystemer brukes for eksempel i kontroll med prosessanlegg på oljeplattformer og i atomreaktorer. Det finnes diagnoseprogrammer, både for medisinske diagnoser og for feilsøking i nettverk.

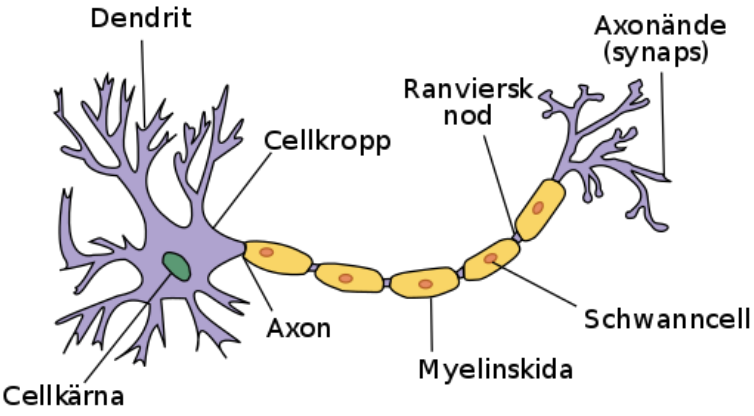
18.3.4 Andre typer kunstig intelligens teknologi

Det finnes også andre måter å lage systemer som etterligner menneskelig resonnering på. Nevrale nett, genetiske algoritmer og fuzzy logikk er de teknologiene det i dag

arbeides mye med. Felles for disse er at de brukes på problemstillinger som er for kompliserte til at mennesker kan håndtere dem innen rimelig tid.

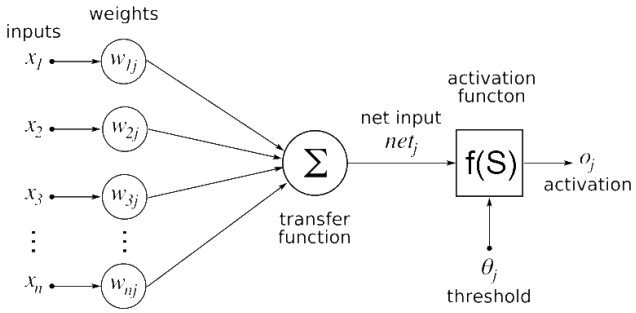
Nevrale nett

Nevrale nett etterligner hjernens struktur med hjerneceller (nevroner) og koblinger mellom disse. Denne etterligningen kan være i selve maskinvaren, ved at en rekke enkeltprosessorer kobles sammen, eller i programvaren, som simulerer en tilsvarende struktur. Det grunnleggende som skjer i hjernen er *parallell prosessering*. Et nevralt nett virker ved at det skjer en massiv parallellprosessering i et stort antall noder. Ingen av nodene kontrollerer systemet som helhet.



Figur 18.2 Et nevron (Illustrasjon: Mehinger, Wikipedia Commons https://commons.wikimedia.org/wiki/File:Neuron_sv.svg)

Et nevralt nett har først og fremst evnen til å lære, og skiller seg dermed fra ekspertsystemene, der kunnskapen nærmest er konservert. En typisk anvendelse av nevrale nett er i mønstergjenkjenning, som er viktig i mange sammenhenger. Nevrale nett er imidlertid ikke uproblematiske å bruke. Til forskjell fra ekspertsystemene kan de ikke alltid forklare hvordan de har kommet frem til et svar, og behøver ikke nødvendigvis gi det samme svaret dersom de prøver på nytt.



Figur 18.3 En modell for et kunstig nevron (Illustrasjon: Geetika Saini, Wikimedia Commons https://commons.wikimedia.org/wiki/File:Artificial_neural_network.png)

Nevralnett brukes i dag i en rekke sammenhenger. Vi skal spesielt merke oss at denne teknologien brukes i programvare for Data Mining, noe vi kommer tilbake til i kapittel 19.

Genetiske algoritmer

Genetiske algoritmer bygger på idéen om evolusjon, altså tilpasning gjennom forandring. Svært forenklet kan vi si at en genetisk algoritme går ut på å lage og prøve ut forskjellige alternativer for så å velge den beste. Også med genetiske algoritmer kan vi oppnå en evne til å lære.

Fuzzy logikk

Ved programmering bruker vi i stor utstrekning logikk for å avgjøre valg. Vanlig logikk er basert på verdiene sant og usant, og det er denne typen logikk som ligger til grunn for det meste av programmeringen, inkludert ekspertsystemer. Fuzzy logikk er en type logikk der vi ikke lenger opererer med sant og usant, men i stedet en hel serie kategorier. Dette er mer i overensstemmelse med hvordan vi oppfatter verden rundt oss. Hvis vi ønsker å uttrykke noe om temperatur, vil de færreste av oss innskrenke oss til bare å operere med verdiene varmt og kaldt. I stedet har vi en hel skala med karakteristikker av temperaturen, som iskaldt, kaldt, kjølig, lunkent, varmt og brennvarmt. Dette er nettopp måten man bruker fuzzy logikk på.

Fuzzy logikk har vist seg å være svært nyttig for å lose komplekse problemer. Vanskeligheten er at det er langt fra enkelt å lage modeller basert på denne typen logikk. Programmering med fuzzy logikk ser for øvrig ut til å være mye mer utbredt i Asia enn i Vesten.

Det finnes i dag T-baner som styres ved hjelp av fuzzy logikk. Algoritmene i programmet brukes til å foreta myk bremsing og akselerasjon av togsettene.

Kapittel 19 Data Mining

Data mining krever, i motsetning til alle de andre analysemetodene vi har sett på, visse kunnskaper i statistikk. Data mining går først og fremst ut på å finne mønstere i store datamengder, og denne søkingen er basert på statistiske metoder og maskinlæring (Hobbs 2003). Typiske spørsmål man søker svar på med data mining er av typen:

- Hvilke kunder er det mest sannsynlig vil kjøpe induksjonskomfyr?
- Hvor i landet vil det være gunstigst å prøve ut et nytt produkt?
- Hvilke kampanjer kan være gunstig å kjøre før påskeferien?
- Hvilke mennesker er mest utsatt for å få høyt blodtrykk?

Data mining kan deles i to kategorier (ibid.):

- Analyse der vi vet hva vi ser etter. Dette kalles *styrt* (directed) læring
- Analyse der vi ikke vet hva vi ser etter. Dette kalles *fri* (undirected) læring

Teknikker for data mining kan klassifiseres på forskjellige måter. Typisk er at man opererer med tre eller fire typer analyser. Vi skal her se på klassifiseringen brukt av Hobbs (ibid.). Disse opererer med tre forskjellige typer analyse:

- Assosiasjon
- Klustering
- Klassifikasjon

Av disse hører de to første til kategorien fri læring, mens den siste hører til styrt læring.

Det er mange forskjellige verktøy tilgjengelig for data mining. Oracle har sine egne verktøy, innebygget i Oracle Enterprise Server. Oracle-verktøyet består av to deler: Oracle Mining Server (metadatabase) og Oracle data mining Java API (et API for programmering av data mining algoritmer). SPSS og SAS er to produsenter av statistikkverktøy som de senere årene har gått inn i data mining markedet. Weka er det mest brukte open source-verktøyet.

19.1 Arbeidsflyt

Arbeidsflyten i en data mining prosess består av flere trinn (ibid.):

1. Forbehandling av data. Dette kan enten gjøres automatisk av databasesystemet (Oracle kan dette) eller spesifiseres av brukeren. En type forbehandling er såkalt ”binning”, som er en gruppering av data etter verdier. En annen type forbehandling er spesifisering av viktigheten av attributter. Dette fordi ikke alle attributter vil ha den samme virkning på det vi vil undersøke. For eksempel har en persons alder en viss betydning for musikksmaken, mens hårfargen neppe har det.
2. Spesifikasjon av data som skal undersøkes. Disse data må plasseres i en enkelt tabell Hver forekomst i datavarehusets tabeller kalles et case, for eksempel

”kunde nr 45, alder 40, mann, lærer, DVD-spiller kjøpt 1/0 2004”. Et slikt case kan representeres enten som en post (transaksjonelt format) eller som multiple poster (ikketransaksjonelt format). Oracle vil automatisk konvertere data til multipell post representasjon.

3. Spesifisere parametere for analyse. Man må nå spesifisere hva slags analysetype man skal bruke (assosiasjon, klustering, klassifikasjon). Videre må man spesifisere hvordan hvert enkelt attributt skal behandles av data mining algoritmene.
4. Bygge en modell. Dette er selve analysealgoritmen som brukes.
5. Bruke modellen på data. Modellen kjøres nå på de forbehandlede data.

Transaksjonelt format				
ID	alder	kjønn	yrke	kjøp
1	40	mann	lærer	DVD
2	35	kvinne	lege	TV
3	50	mann	lektor	CD

Ikketransaksjonelt format		
ID	attributt	verdi
1	alder	40
1	kjønn	Mann
1	yrke	lærer
1	kjøp	DVD
2	alder	35

Figur 19.1 Tabellformater for data mining (etter Tierney 2014)

19.2 Assosiasjon

Assosiasjon vil si at man finner objekter som ofte hører sammen. Et banalt eksempel er pølser og rundstykker, eller pizza og cola. Assosiasjon brukes av nettbutikker, spesielt Amazon, som setter opp forslag på bøker og andre artikler både ut fra dine tidligere kjøp og hvilke kjøp andre kunder har gjort. Et eksempel er at mange av de kundene som kjøper Tolkiens ”Ringenes herre” også kjøper ”Hobitten”. Ikke fullt så mange vil også kjøpe Silmarillion, stadig av samme forfatter. Mange av de som kjøper ringenes herre vil også kjøpe Narnia-bøkene. På grunnlag av dette settes automatisk opp anbefalinger for den enkelte kunde.

Assosiasjonsanalyser kalles også ”market basket”-analyser. Dette er typisk analyser utført for markedsføringsformål. Data maning brukt av bedrifter er typisk noe markedsførerne driver med.

Det er ikke nok bare å finne en sammenheng mellom to eller flere objekter. I tillegg må man også ha frekvenser for hvor hyppig de opptrer sammen. Dette kan beregnes på to forskjellige måter:

- hvor ofte to objekter opptrer sammen, f. eks. hvor ofte en kunde kjøper pizza og cola sammen
- hvor ofte forekomst av et bestemt objekt også har forekomst av et bestemt annet objekt, for eksempel hvor mange prosent av kjøpene av pizza som også har kjøp av cola.

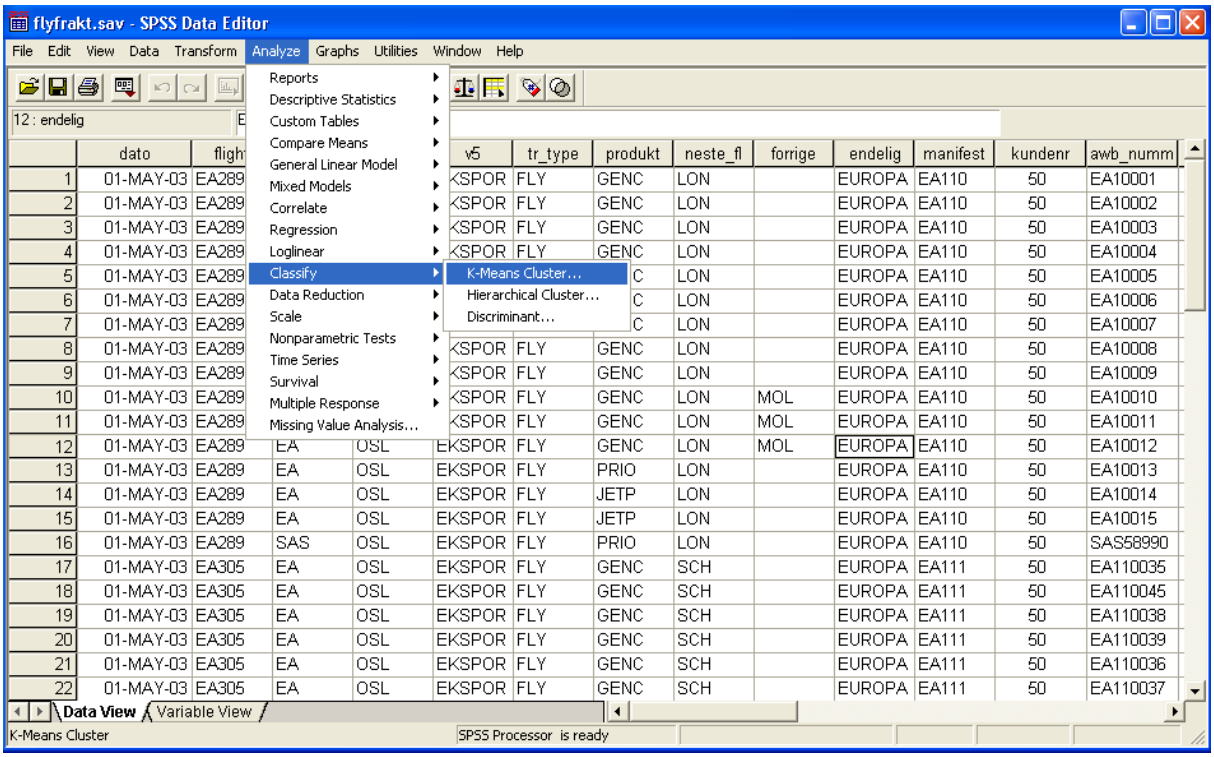
19.3 Klustering

Klustering er en metode der store datamengder komprimeres til et mindre antall grupper eller klustere. Poenget med data mining er at slike grupper ikke alltid er lette å "se" for oss, men data mining algoritmer kan finne dem. Klustering har vært brukt i mange år innen forskning, spesielt innen medisin. Et eksempel er analyse av befolkningsdata, der man kan finne ut at visse grupper har hyppigere forekomst av spesielle sykdommer enn andre grupper. Slike analyser ble gjort allerede på 1800-tallet, da man fant sammenheng mellom sykdommer som kolera og forurensede vannkilder.

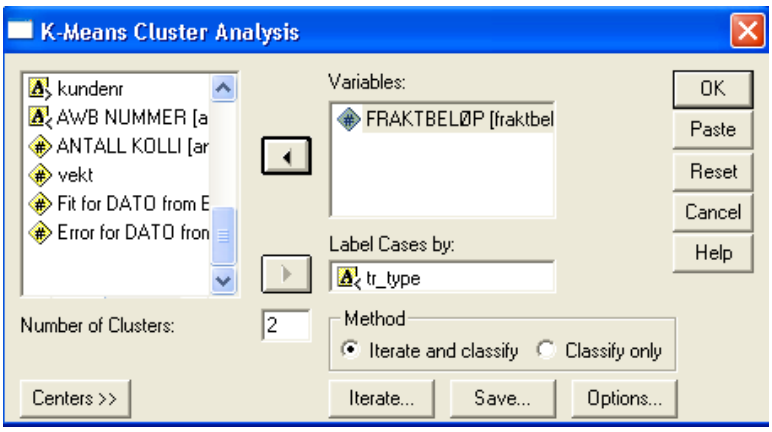
Klustering brukes også mye i markedsanalyser, der man ønsker å finne spesifikke segmenter i markedet. Man søker da å dele inn kundene etter forskjellige kriterier. Eksempler på slike segmenter, eller klustere, kan være "mann mellom 30 og 50 år, med høyere utdannelse og ansatt i næringslivet, bosatt på Østlandet". Denne gruppen vil være temmelig forskjellig fra "kvinne, alder over 50 år, uten utdanning ut over obligatorisk grunnskole, hjemmeværende og bosatt i Midt-Norge".

SPSS har de senere årene siktet seg inn mot data mining-markedet. Med SPSS kan man foreta en rekke statistiske analyser på data, men man kan også kjøre fullverdige data mining analyser. Maskinlæring utføres av en egen data mining modul basert på nevralnetteknologi, men basisinstallasjonene av SPSS kan utføre en god del enklere analyser. Nedenfor er vist SPSS med valg av klustering.

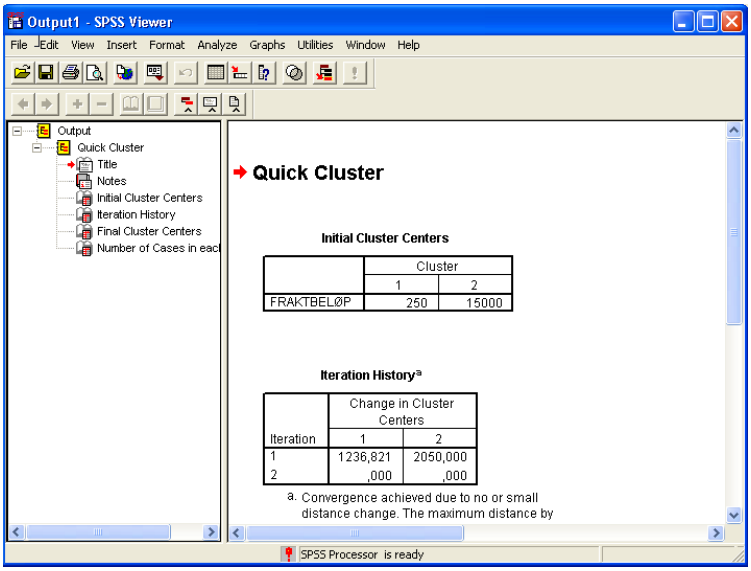
En typisk algoritme som brukes både av Oracle og SPSS er k-gjennomsnitt. Denne grupperer elementer basert på hvor store forskjellene er mellom dem.



Figur 19.1 Klustringanalyse i SPSS



Figur 19.2 Spesifiseringer for k-gjennomsnitt analyse med SPSS



Figur 19.3 Output fra klustringanalyse i SPSS

19.4 Klassifikasjon

Et eksempel på klassifikasjon kan være at man driver en forretning for elektronikk og ønsker å vite hvilke kunder som kan tenkes å kjøpe et nytt produkt. Fra markedsføringsfaget vet vi at det er viktig å identifisere hvem man skal drive målrettet markedsføring mot, siden man ofte kan ta en høyere pris fra disse og dermed raskere tjene inn investeringer man har gjort.. Forskjellen på klassifikasjon og klustering er at man med klustering ikke vet hvilke grupper man har, mens man med klassifisering gjør det. I vårt eksempel har man to grupper kunder: de som kan tenkes å kjøpe OLED-TV og de som ikke vil det. Dette er et eksempel på styrt læring, mens klustering er fri læring.

Typiske algoritmer for klassifikasjon er Bayes-algoritmer. Det er i hovedsak to forskjellige algoritmer som brukes: Naïve Bayes og Adaptive Bayes Network. Den siste bruker et beslutningstre under analysen.

Kapittel 20 Big Data

Big Data har de siste årene seilt opp som et sentralt «buzzword». Noen forsøker å innføre begrepet «Stordata» på norsk, men det ser ikke ut til å få noe gjennomslag. Praktisk talt all litteratur om emnet er tross alt på engelsk. Vi skal her se på hva dette dreier seg om.

20.1 Hva karakteriserer Big Data?

Big Data handler om innsamling, analyse og presentasjon av svært store datamengder, der både størrelsen og kompleksiteten gjør at analysemetodene vi ellers kjenner fra Business Intelligence ikke er brukbare. Slik sett kan vi si at det dreier seg om en utvidelse av Business Intelligence. Big Data dreier seg imidlertid om mer enn dette: data er ofte ustrukturerte (som tekst og bilder), og innsamlingen og analysen skjer ofte kontinuerlig.

Noen eksempler på hvor ekstremt store datamengder genereres er:

- Sosiale medier som Facebook og Twitter
- Søkemotorer som Google
- Internettbruk
- Meteorologiske data
- Forskningsdata, som ved Large Haldron Collider og gensekvensering
- Logistikkdata
- Transportdata
- Bruk av radio og TV
- Aksjehandel
- Strømforsyning

Big Data hører sammen med the Internet of Things. Dette begrepet bygger på at en rekke enheter er utstyrt med chips som genererer data, og disse er koblet til internett. Det kan dermed samles inn store datamengder fra alle disse enhetene. Disse dataene analyseres ofte kontinuerlig, det vi kaller i real-time.

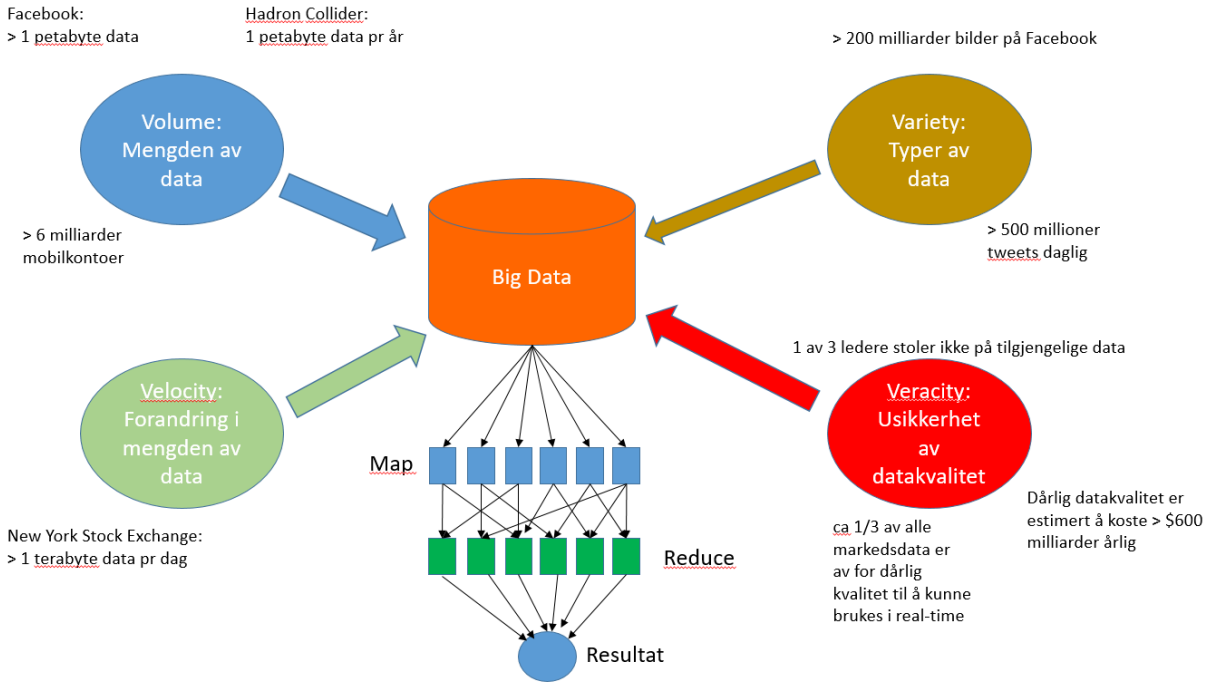
I 2001 kom en rapport fra det som i dag heter Gartner Group, der det ble pekt på utfordringene med store datamengder. Rapporten definerte en modell for store datamengder basert på «de tre V'er»: Volume, Velocity og Variety. Senere har mange lagt til en fjerde «V»: Veracity. Vi skal forklare disse begrepene:

- Volume: Big Data observerer og analyserer store datamengder
- Velocity: Data endres kontinuerlig, og analyseres ofte i real-time
- Variety: Analyserer tekst, bilder, lyd og film, i det hele tatt det vi gjerne omtaler som *ustrukturerte data*
- Veracity: Kvalitetsforskjeller innen datamengdene

I et tradisjonelt datavarehus lagres typisk *strukturerte* data, altså det vi finner i tabellene i en relasjonsdatabase.

For analyse av svært store datamengder brukes parallellprosessering. Analysearbeidet deles opp i deler som fordeles til et (ofte stort) antall prosessorer. Typisk er også at prosessene bruker egne filsystemer i stedet for en relasjonsdatabase.

Nedenfor er vist en modell for Big Data basert på «de fire V'er»:



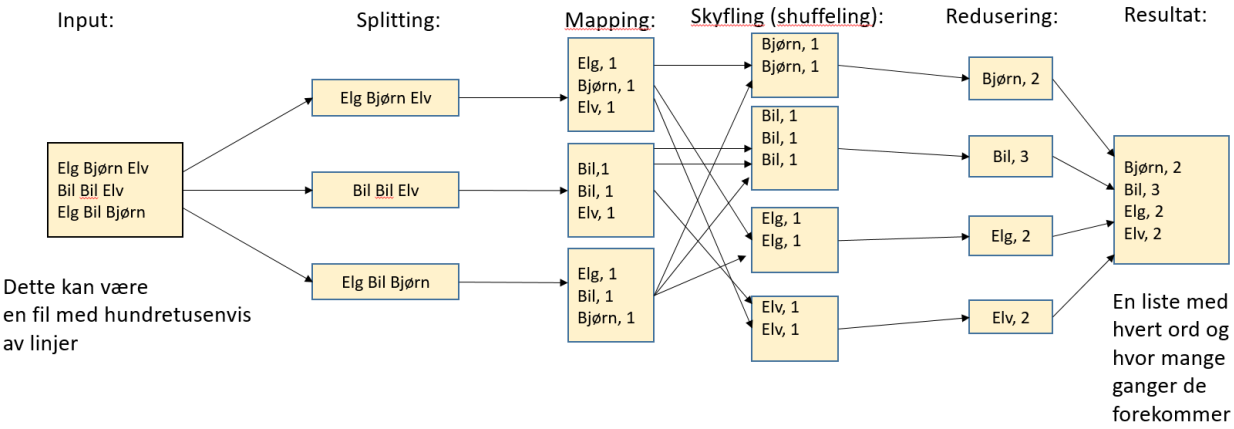
Figur 20.1 Big Data
Etter https://tomyrhymond.wordpress.com/category/technology/

Mange tar også med en femte «V»: Verdi. Analysen skal føre til en verdi i en eller annen form, typisk å gi et bedre beslutningsgrunnlag.

20.2 MapReduce-prosessen

Sentralt i behandlingen av store mengder ligger en algoritme som kalles MapReduce. Denne består av tre steg: mapping, omflytting og redusering.

For å illustrere mapReduce-prosesser brukes ofte eksempelet nedenfor. Her er utgangspunktet tekst, for eksempel i form av tweets, Facebook-innlegg, bøker m.m., og resultatet skal være en opptelling av antall ganger hvert ord brukes:



Figur 20.2 Prinsippet bak map-reduce

Det som her har skjedd, er at ustrukturerte data, gjennom mapping og reduksjon, er blitt forvandlet til strukturerte data.

Prosesseringen skjer ikke bare på en prosessor, med fordeles på mange prosessorer. Mange datamaskiner har i dag en flerprosessorarkitektur, men man kan også koble sammen flere maskiner. Dette blir omtalt i neste kapittel.

20.3 Visualisering

Visualisering er en viktig del av Big Data. De store datamengdene gjør at man i større grad enn med datavarehus bruker grafiske presentasjoner. Nedenfor er en oversikt over forskjellige typer visualisering:

- Diagrammer
- Ordskyer
- Kart

Altså mye av det samme som vi har brukt Power BI til.

Litteratur

Agosta 2000:
Agosta, Lou: The Essential Guide to Data Warehousing
Prentice Hall PTR, New Jersey, 2000

Alter 2002:
Alter, Stephen: Information Systems. The Foundation of E-Business
Prentice Hall 2002

Anahory 1997:
Anahory, Sam og Dennis Murray: Data Warehousing in the Real World
Harlow: Addison-Wesley 1997

Busch 1995:
Busch, Tor og Jan Ove Vanebo:
Organisasjon, ledelse og motivasjon
3. utgave, TANO 1995

Christensen 1999:
Christensen, Gunnar E., Stein Erik Grønland, Leif B. Methlie:
Informasjonsteknologi: strategi, organisasjon, styring
Cappelen akademisk 1999

Connolly 2014:
Connolly, Thomas og Carolyn Begg: Database Systems,
Sixth Edition, Addison-Wesley 2014

Curtis 2002:
Curtis, Graham og David Cobham:
Business Information Systems. Analysis, Design and Practice
4th ed., Prentice Hall 2002

Devlin 1988:
Devlin, B. og P. Murphy: An Architecture for a Business Intelligence System.
IBM Systems Journal 27, no 1, pp. 60 - 80

Dodge 2000:
Dodge, Gary og Tim Gorman: Essential Oracle8 Data Warehousing
New York: John Wiley & Sons, 2000

Giovinazzo 2000:
Giovinazzo, William A.: Object-Oriented Data Warehouse Design. Building a Star Schema. Prentice-Hall PTR, New Jersey 2000

Gottschalk 2002 - 1:
Gottschalk, Petter: Informasjonsledelse. Fra forretningsbehov til informasjonssystem. Universitetsforlaget 2002

Gottschalk 2002 – 2:
Gottschalk, Petter: IT-strategi
Fagbokforlaget, Bergen 2002

Haraldsen 1998:
Haraldsen, Arild: IT på norsk. Strategisk bruk av IT.
Oslo: Tano Aschehoug 1998

Hatch 2007:
Hatch, David: Smart Decisions: The Role of Key Performance Indicators. Intelligent Enterprise, 5. November 2007

Heie 2003:
Heie, Tor Henrik: Problemstillinger i datavarehus og system utvikling.
Presentasjon for studentene ved Høgskolen i Buskerud, 2003

Hobbs 2003:
Hobbs, Lilian, Susan Wilson, Shilpa Lawande:
Oracle 9iR2 Data Warehousing
Digital Press, Elsevier 2003

Hoffer 2002:
Hoffer, Jeffrey A., Mary B. Prescott, Fred R. McFadden: Modern Database Management 6th edition, Pearson Education International, 2002

Inmon 1996:
Inmon, W.H: Building the Data Warehouse
2. ed, John Wiley & Sons, New York 1996

Inmon 1997:
Inmon, W.H.: Iterative Development in the Data Warehouse
DM Review 7 (November), pp. 16 – 17

Inmon 1998:

Inmon, W.H.: Data Mart Does Not Equal Data Warehouse
DM Review, may 1998

Inmon 2000:
Inmon, W.H.: The Problem with Dimensional Modeling
DM Review, may 2000, pp. 68 – 70

Jessup 2008:
Leonard Jessup & Joseph Valacich: Information Systems Today. Managing the Digital Firm.
Prentice Hall 2008

Jones 2004:
Jones, Gareth R.: Organizational Theory, Design and Change. Text and Cases.
4th ed., Pearson Education International, New Jersey, 2004

Kaplan 1992:
Kaplan, Robert S. og David P. Norton: The Balanced Scorecard: measures that drive performance. Harvard Business Review Jan – Feb pp. 71 – 80

Kaplan 1996:
Kaplan, Robert S. og David P. Norton: The Balanced Scorecard. Harvard Business School Press, Boston.

Kimball 2013:
Kimball, Ralph: The Data Warehouse Lifecycle Toolkit. The Definitive Guide to Dimentional Modeling
John Wiley & Sons, 1998

Kristensen 1997:
Kristensen, Terje:
Nevrale nettverk, fuzzy logikk og genetiske algoritmer
Cappelen Akademisk Forlag, Oslo 1997

Laudon 2017:
Kenneth C. Laudon og Jane P. Laudon: Manage Information Systems
15th ed., Pearson Prentice Hall International Editions, 2017

Lindsted:
Lindsted, Dan: data vault overview: the next evolution in data modelling
http://danlinstedt.com/

Monk 2007:

Monk, Ellen og Bret Wagner: Concepts in Enterprise Resource Planning
Course Technology, 3rd edition, 2007

Norstella/Norsk EDIPRO 2000:
Norstella/Norsk EDIPRO:
Veiledning i bruk av EDIFACT i ELEKTRONISK SAMHANDLING.
Hefte 1: En innføring i grunnleggende begreper og teknologier.
Norsk EDIPRO, Oslo 2000

Poe 1998:
Poe, Vidette, Patricia Klauer, Stephen Brobst:
Building a Data Warehouse for Decision Support
2ed, Prentice Hall PTR, New Jersey 1998

Porter 1980:
Porter, Michael E.: Competitive Strategy: Techniques for Analyzing Industries and Competitors.
New York: Free Press, 1980

Porter 1985:
Porter, Michael E.: Competitive Advantage: Creating and Sustaining Superior Performance.
New York: Free Press, 1985

Reve 1989:
Reve, Torgeir og Kjell Grønhaug: Strategi og organisasjon.
TANO, Oslo 1989

Sethi 1998:
Sethi, Vikram og William R. King:
Organizational Transformation Through Business Process Reengineering. Applying the Lessons Learned. Prentice Hall 1998.

Sperley 1999:
Sperley, Eric: The Enterprise Data Warehouse. Planning, building and Implementation.
Prentice-Hall 1999

Turban 2007:
Turban, Efraim, Jay E. Aronsom, Ting-Peng Liang, Ramesh Sharda:
Decision Support and Business Intelligence Systems. Pearson Prentice Hall, New Jerset 2007

Sharda 2014:
Sharda, Ramesh; Dursun Delen, Efraim Turban:

Business Intelligence and Analytics. Systems for Decision Support.
Pearson Global Edition 2014

Tierney 2014:
Tierney, Brendan: Predictive Analysis with Oracle Data Miner.
McGraw-Hill Education, 2014

Skriftserien nr. 47
2020
—

Business Intelligence og datavarehus
En praktisk tilnærming

—

Forfattere: Trond R. Braadland

—

ISBN 978-82-7860-434-2
ISSN 2535-5325

usn.no