## Individuell inlämning 4

**Datalager**  
För att ett datalager ska kunna fatta beslut effektivt har fyra olika karaktärer tagits fram:

* *Ämnesinriktat*Data som befinner sig i samma ämnesområde ska kopplas ihop
* *Icke-flyktig*Data ändras inte när den väl är lagrad
* *Integrerad*Data som inte har med varandra att göra (av olika typ och olika ämnesområden) ska ändå vara konsekvent
* *Tidsvarierande*Data ska vara kopplad till en viss tidsperiod

Datalager är alltså en samling av data som lagras på sådant sätt att den uppfyller de olika karakteristiska kraven ovan, och som därefter används primärt för att fatta beslut i en organisation. En organisation samlar alltså på sig data i sitt datalager som kan bidra till fördelaktiga beslut i framtiden.

**Operativa databaser mot datalager**De traditionella operativa databaser stödjer *Online Transaction Processing (OLTP)*. På grund av sin karaktär där de arbetar med transaktioner kan de endast hantera en begränsad mängd sådana - en transaktioner gör sitt, sen är det nästa transaktions tur. De flesta operativa databaser är objektorienterade- eller relationsdatabaser och byggdes för att tjäna slutanvändares behov. Alla uppdateringar, insättningar och borttagningar bör ske kvickt i realtid för att den ska kunna representera nuläget i en organisation så bra som möjligt.

På andra ändan är datalager som stödjer *Online Analytical Processing (OLAP)*. Detta innebär att ett datalager arbetar med hjälp av en mängd operativa databaser där data hämtas från, men kan också innehålla data såsom metadata eller väldigt översiktligt och sammanfattande data. Data passerar inte lika febrilt som i en operativ databas, utan snarare periodiskt.  
  
**ETL**

*Extraction, transformation* och *loading* systemet är en del av *datalagrets arkitektur* vars syfte är att hämta data från olika källor (s.k. *data stores*), transformera och integrerar den. ETL-systemet används också för att bekräfta integritet och validitet hos data (s.k. *data cleaning*) innan den laddas in i datalagret.   
  
**Data marts**

Vissa delar av ett datalagers data samlas ihop till ett sorts subset som kallas *data mart*. Detta subset ska vara i ett visst ämnesområde och brukar rikta sig åt en specifik målgrupp inom en organisation.

**Hur ett datalager utvecklas**

Det finns främst två separata metoder för att utveckla ett datalager. *Top-down* och *bottom-up*. Top-down grundar sig i att det är omöjligt att determinera hur ett system kommer belastas i en organisation, och design- och utvecklingsstadie ska gå så fort som möjligt. Användare i första användningsiterationen ger feedback.  
*Bottom-up* grundar sig att designa olika former av *data marts* istället för att hastigt utveckla en hel databas direkt. Strategin är alltså att successivt skapa data marts för hela organisationen.  
Den notabla skillnaden mellan metoderna är att bottom-up brukar ses som en lättare implementering - snabbare *Return on Investment*.

**Olika typer av OLAP**

Det finns olika typer av OLAP. De två mest utbredda typerna är *Relational OLAP* och Multidimensional *OLAP*.

* ROLAP   
  Arbetar direkt med relationsdatabaser och byggs upp med ER-modellen. Undertyper till ROLAP är *star schema-* och *snowflake-schema*. Star-schema går ut på att ha en central tabell som kallas *faktatabeller* som lagras både nycklar och data. Nycklarna kan ses som *foreign keys* till andra tabeller. Värt att notera att i star schema får endast denna centrala faktatabell ha nycklar till tabeller runt. Det är just detta som skiljer star-schema från snowflake-schema - ett snowflake-schema får ha iterativa “faktatabeller”.
* MOLAP   
  Bygger på en sorts datakub där kubens *dimensioner* kan ses som att de representerar kolumner i en relationsmodell. Syftet med att modellera data på detta sätt är för att kunna hämta data från utvalda *dimensioner*. Om x och y representerar butik och säljsumma, och z representerar vilken månad kan en s.k. *slicing* göras för för att få t.ex. säljstatistiken för oktober. Det går att göra på vilken dimension som helst. Kuben kan även roteras (*pivoted*) för att via vilken tvådimensionell vy som önskas. Vidare kallas det *dicing* ifall en tredimensionell kub skärs två gånger (då den figurativt bildar en tärning av kuben).

**Vyer för ett datalager och materialisering**

Precis som en operativ databas använder datalager sig också av *vyer* för att skapa skräddarsydda perspektiv av data. Däremot på grund av ett datalagers natur att lagra enorma mängder data kan skapandet av vyer ta orimligt lång tid. Ett alternativ till att skapa vyer just när de behövs är att *materialisera* dem. Detta innebär att förberäkna vyerna och lagra resultatet för senare användning.   
  
Det finns olika policies för hur materialisering sker såsom:

* *Lazy refresh*  
  En vy uppdateras när en förfrågan inom vyn sker och vyn är obsolet
* *Periodic refresh*En vy uppdateras periodiskt, oavsett användning av vyn
* *Forces refresh*En vy uppdateras efter ett visst antal uppdatering skett till vyns underliggande tabeller.

**Data mining**

Data mining syftar på att utvinna värdefull och ny information från en enorm mängd data. För att data mining ska vara effektiv och lönsam krävs en växande mängd data och lagring av historisk data. Ofta används ett datalager som källa till data mining då den redan innehåller en enorm mängd data som ofta till och med är integritet- och validitetkontrollerat (genom ETL-system som beskrevs tidigare).   
  
Data mining används alltså för att studera data utan att först formulera någon hypotes alls. Utan med data mining hittas mönster genom att studera data som helhet - mönster som troligen inte skulle hittas annars.  
  
Vanligen sker data mining endast på s.k. *flat files* som innebär att alla värden i en databas finns i en enda enhet (eng. *record*). Detta skiljer sig då mot relationstabeller som normaliseras. Vidare ska data vara *numerisk*, såsom heltal, eller *kategorisk*, såsom man eller kvinna. Data mining kräver alltså att data ska lagras på ett visst sätt, annars krävs omformatering, något som kan ta lång tid när det gäller så stora mängder data.  
  
**CRISP-DM**

*Cross Industry Standard Model for Data Mining* är en modell för att ta sig igenom alla steg som finns i data mining. Modellen går ut på att först få förståelse i företaget eller organisationen om krav och mål, skapa förståelse för tillgänglig data, välja modell för att få ut så bra resultat som möjligt, testa och evaluera, samt implementera.

**Regression**  
Data mining går grund och botten ut på statistikens *regression* - förutspå en viss variabel som en mängd av determinanter {x\_1, x\_2 … x\_n} determinerar en mängd beroende variabler {y\_1, y\_2 … y\_n}. Regression handlar alltså om att hitta vad som bidrar till vad eller vad som korrelerar med vad, och därmed kunna förutspå värden.   
  
**Beslutsträd**

För att kunna klassificera data är att använda *beslutsträd*. Detta går ut på att skapa olika förgreningar beroende vilket värde som stoppades in i toppen. En gren i ett träd kan sedan förgrenas igen, sådant att ett värde som stoppas in i toppen följer en viss förgrening ända ner till ett *löv* där den helt och hållet enligt beslutträdets mening fått en klassifikation. Varje förgrening är ofta olika meriter som input-variabeln matchas mot. T.ex. resultaten olika prov för en student kan användas för att klassificera denna till dålig, bra eller utmärkt student.

**Neural networks**

Ett sorts relationsnätverk ämnad åt att efterlikna ett biologiska neuronnät. Tekniken för att skapa ett neural network är leta efter olika starka samband mellan olika data - en sorts generaliserad form av regression. Det som gör denna teknik unik är att den är byggd för att lära sig och adaptera till ny information.

**Användningsområden för data mining**

Då data minings definition kan brytas ner till att “Hitta hur information kopplas till annan information så att värden kan förutspås” är användningsområdet brett. T.ex. hitta en kund som är mest trolig att köpa en produkt efter reklam visats eller att en bank förutspår om en viss person troligen kommer begå bedrägeri.

**Big data**

*Big data* syftar på att samla in en så stor mängd data att den inte kan hanteras av traditionella databaser. För att definiera big data används ofta de fem V.

* *Volume*  
  Syftar på kvantitet av data som samlas in. Detta sker från t.ex. sensorer, skärmar, sociala medier och loggfiler.
* *Variety*  
  Syftar på de olika typer av data som samlas in. Vanligen handlar det om textdata såsom kommentarer eller inlägg i social medier, men även övervakningskamerorna inspelningar.
* *Velocity*  
  Syftar på att data hela tiden är i rörelse. Alltså data skickas runt hela tiden mellan t.ex. datornätverk, sensorer eller mobiltelefoner.
* *Veracity*  
  Syftar på noggrannheten på noggrannheten i data. Data kanske blir korrupt under olika former av överföringar, såsom vid videoströmmar.
* *Value*  
  Syntar på värdet och fördelarna som en organisation kan få en genom användning av big data. Denna läggs ofta som överlappande av de tidigare fyra V.

**Hadoop**

För att hantera stora filer används systemet *Hadoop*. Hadoop-systemet byggs upp av två komponenter:

* Filsystemet *HDFS*  
  *Hadoop Distributed File System* är en batch orienterad typ av DFS (Distributed File System) sådant att det kan hantera dataset som skulle vara för stora att lagra på enskild maskin. HDFS fördel specifikt är att det kan dela upp stora filer i mindre komponenter, *blocks*, som bidrar till en god felhantering och feltolerans - varje *block* kan hantera individuellt. Dessa block placeras sedan ut över flera s.k. *nodes* som i Hadoops mening är en enskild maskin.
* *MapReduce Parallel Programming Paradigm*  
  Block-modellen som HDFS är grundad på bidrar också till möjligheten att parallellisera jobb som väntar. Om olika jobb behöver komma åt olika delar av den stora mängden data kan dessa jobb köra parallellt på varsitt block. Namnet *MapReduce* är två delar:
  + *Map*  
    Filtrerar och/eller transformerar input data på sådant sätt att nästa steg (*Reduce*) kan arbetat effektivare.
  + *Reduce*

Utföra någon form av beräkning eller aggregation av data som kom in genom föregående steg (*Map*).

**Hive och HiveQL**

*Hive* bygger på Hadoop med traditionella databasförfrågningar och ger möjlighet att översätta data lagrad i Hadoop till tabeller. Hive kommer därmed tillsammans med *HiveQL*, *Hive Query Language*, som är ett SQL-interface som möjliggör SQL-förfrågningar direkt på data i Hadoop.

Vidare organiserar Hive sin data i databaser, tabeller, *partitioner* och *buckets*. Databas i denna mening kan ses som en den högsta abstraktioner och tabeller följer samma princip som i relationsmodellen. Däremot är *partitioner* och *buckets* unika för Hive.

* Partitioner  
  Ett sätt att organisera tabeller enligt ett värde av en viss kolumn - partition kolumnen.
* Bucket  
  Ett sätt att tillhandahålla både tabeller och partitioner via hash-värden på en viss kolumn. Det är alltså ett sätt vidare dela in data sådant att det snabbt går att hämta ju denna data. Något som är användbart vid t.ex. *join*.

**NoSQL**

Genom åren har det visat sig att relationsdatabaser varit svaga eller opassande inom vissa områden. Nackdelar såsom:

* Databasscheman måste definieras i designfasen, innan någon data kommit in (annars kan inte relationsdatabasen lagra denna data)
* Relativt dålig skalbarhet och flexibilitet
* ACID-transaktioner bidrar i vissa fall till en del prestandaproblem

NoSQL system var därför designade för att lösa problemet med relationsdatabaser. Viktigt att påpeka är att NoSQL inte ersätter relationsdatabaser, utan endast i vissa områden är det mer passande att använda NoSQL system, såsom när enorma mängder data ska hanteras.  
Fördelarna med NoSQL är bland annat: Hantera enorma mängder data i realtid, flexiblare databasscheman och parallell bearbetning (använder Hadoop-teknologier som beskrev ovan).

**Olika typer av NoSQL system**

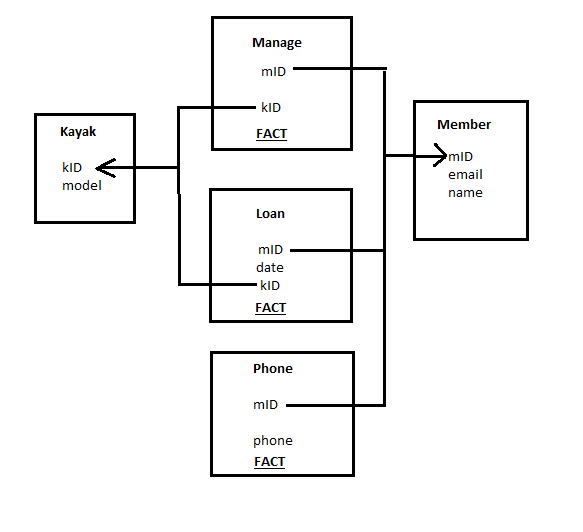
* *Key-value pair*  
  Bygger på HDFS på sådant sätt att värden kan identifieras med en nyckel. Nyckeln fungerar som ett sorts index och efterliknar hur datorminne fungerar.
* *Column-oriented table*Bygger på Googles *BigTable* och arbetar runt kolumner. Kolumner kan grupperas till s.k. *kolumn-familjer* som kan dynamiskt växa eller minska.
* *Document-oriented*  
  Bygger på att all data kan kapslas in i ett *dokument* som i denna mening syftar på en enhet med data med stor flexibilitet. En dokumentet kan jämföras med rader i en relationsdatabas men kan variera i för vilka typer av värden den innehåller. Detta brukar vara i format som XML.
* *Graph-oriented*Bygger på att data behandlar som om det vore antingen en hierarkisk- eller nätverksstruktur, dvs. noder kopplade till varandra och nodkorsning (eng: *node traversal*) behöver utföras för att komma till en viss nod av intresse.

**NewSQL**

Ett sorts hybridsystem mellan relationsdatabaser och NoSQL som kommit på senare tid. Den offrar relations databasens *atomicity* för att få NoSQL prestanda och skalbarhet. NewSQL offrar dock inte hela ACID-principen, men behöver ändå skalbarhet. Detta har visat sig vara passande för t.ex. banksystem.

Individuell uppgift

**Hur skulle man kunna göra ett Star schema utifrån uppgiften ?**



**Vilka intressanta analys(data mining) frågor ser du i uppgiften ?**

* (Om reklam satts upp för kajakklubben) Fungerar reklamen på tänkt sätt? Dvs. är åldersgruppen korrekt? Bör reklamen ändras för att ändra outcome?
* Brukar någon person låna kajak/-er för flera personer? Kan det vara familj eller vänner han brukar komma med? Någon reklam som passar detta?
* Vilka personer är det som är mest sannolika att ha sönder en kajak?
* Vilka personer är det som är mest sannolika att inte betala lånekostnad m.m.?

**Hur skulle en översättning av uppgiften bli till en key value store, respektive till dokumentbaserad databas ?**

**Key-value store:**

Loans

| mID key | value |
| --- | --- |
| 2 | 23,  2020-08-21 |
| 5 | 42,  2020-08-21 |

Manages

| mID key | kID value |
| --- | --- |
| 2 | 24 |
| 6 | 41 |

Members

| mID key | value |
| --- | --- |
| 4 | [ari@gmail.com](mailto:ari@gmail.com),  Ari Svensson |
| 8 | [johan@gmail.com](mailto:johan@gmail.com),  Johan Svensson |

Kayaks

| kID key | value |
| --- | --- |
| 6 | BZ1024 |
| 1 | HK8024 |

Phones

| mID key | value |
| --- | --- |
| 5 | 08-11111111 |
| 1 | 08-22222222 |

**Dokumentbaserad:**  
<Kayak>

<kID>23</kID>

<model>HZ21</model>

</Kayak>

<Kayak>

<kID>26</kID>

<model>HZ23</model>

</Kayak>

<Member>

<mID>23</mID>

<model>HZ21</model>

<phone type=”Private”>08-11111111</phone>

<phone type=”Work”>08-22222222</phone>

<loans>

<loan>

<kid>23</kid>

<date>2020-08-20</date>

</loan>

<loan>

<kid>24</kid>

<date>2021-04-20</date>

</loan>

</loans>

<manages>  
 <kID>26</kID>

</manages>

</Member>