

Databaskonstruktion

Aliens vs. PUCKO

wwwlab.iki.his.se/~a11emmjo/db-konstruktion/PUCKO/index.html

Emma Jonsson

Databaskonstruktion
HT 2012

Innehåll

1	Inledning	4
2	Förberedelser inför implementeringen	4
2.1	Relationsdatamodellering	4
2.2	Vem skulle använda applikationerna och till vad.....	5
2.2.1	Telefonisten.....	5
2.2.2	Fältagenten	5
2.2.3	Administratören.....	5
2.3	Generell plan för applikationen	5
3	Databasen	6
3.1	Tabeller och optimering	6
3.1.1	Datatypsoptimering	6
3.1.2	Restriktionsoptimering	6
3.1.3	Denormaliseringsoptimering	7
3.1.4	Kod-tabeller.....	7
3.1.5	Automatiserade värden.....	7
3.2	Vyer.....	7
3.3	Index	8
3.4	Triggers	8
3.5	Procedurer	8
3.5.1	Hemligstämpla en ras	9
3.5.2	Radera en alien	10
4	PHP-applikationen	10
4.1	Uppdatering av databasen	10
4.2	Exekvering av en procedur.....	12
4.3	Sökning	13
4.4	Kontrasten mellan PDO i PHP och formulär i .NET.....	15
5	.NET-applikationen.....	15
5.1	Paneler.....	15
5.1.1	Välkomstpanelen	15
5.1.2	Synliggöra och dölja paneler.....	15
5.2	Exekvering av en procedur.....	16
5.2.1	Formuläret	16
5.2.2	Funktionen.....	16
5.2.3	Resultatspanelen.....	17
5.3	Uppdatering av databasen	17
5.3.1	Formuläret	17
5.3.2	Funktionen.....	17
5.3.3	Resultatspanelen.....	18
5.4	Sökning	18
5.4.1	Formuläret	18
5.4.2	Funktionen.....	18

5.4.3	Resultatspanelen.....	18
5.5	Kontrasten mellan PDO i PHP och formulär i .NET.....	19
5.6	Att arbeta med ADO	19
6	Reflektioner.....	19
6.1	Tankar om mitt arbete	19
6.2	Tankar om kursen.....	19
6.3	Interaktiva tekniker	19

1 Inledning

Denna rapport sammanfattar resultatet av en uppgift i kursen Databaskonstruktion som gick ut på att implementera en prototyp utifrån ett fiktivt scenario där ett antal systemutvecklingskonsulter arbetat fram en kravspecifikation till en databas för organisationen P.U.C.K.O. (Polismyndigheten för Undanhållandet av Centrala Kunskaper Om utomjordingar). Prototypen skulle framställas i programmeringsspråken SQL, PHP och .NET och arbetet med detta beskrivs i tur och ordning i följande kapitel. Rapporten avslutas sedan med egna reflektioner som uppkommit under arbetets gång.

2 Förberedelser inför implementeringen

Innan arbetet med prototypen påbörjades så gjordes en avgränsning i den ursprungliga kravspecifikationen, dels för att arbetet inte skulle bli allt för omfattande och dels för att poängen med en prototyp ju är att den inte behöver vara fullständig. Den ursprungliga kravspecifikationen bestod av både en ER-modell och en tillhörande text som beskrev hela 26 entiteter och deras relationer, men detta begränsades alltså till följande sex entiteter: Alien, Ras, Registrerad Alien, Registrerad Alien, Skepp och Vapen (se Bild 1).

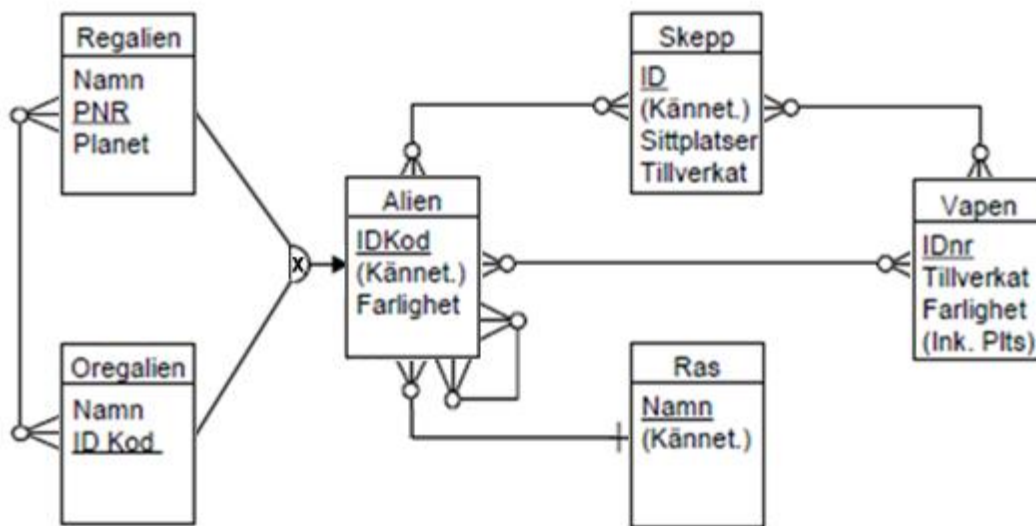


Bild 1 Avgränsningen

2.1 Relationsdatamodellering

För att underlätta för den kommande implementationen i SQL så skapades en relationsdatamodell över den avgränsade ER-modellen (se Bild 1). Detta förtydligade hur tabellerna skulle se ut i databasen och vilka relationer som krävde egna tabeller för sin funktion. Slutresultatet blev enligt följande:

```
planet(nr, namn)
skepp (skeppID, sittPlatser, tillverkningsPlanet )
ras (nr, namn)
farlighet(nr, typ)
alien (alienID, namn, farlighet , ras )
oRegAlien ( alienID , idKod)
regAlien ( alienID , pNr, planet )
vapen (vapenID, tillverkningsPlanet , farlighet )
```

vapenAgarAlien(vapen , alien)
vapenAgarSkepp(vapen , skepp)
butiker(nr, namn, planet)
vapenButiker (vapenID , butik)
skeppsCrew (skeppID , alienID)
bekantskap (alien1 , alien2)
omReg (oregAlien , idKod , regAlien , pNr)
skeppKannetecken (skeppID , tecken)
rasKannetecken (rasNamn , tecken)
alienKannetecken (alienID , tecken)

2.2 Vem skulle använda applikationerna och till vad

Tanken var ju att applikationerna skulle användas inom verksamheten P.U.C.K.O. men mer exakt så skulle den avgränsning som gjorts användas i de följande tänkta scenarion som beskrivs i detta kapitel.

2.2.1 Telefonisten

En telefonist, även kallad utomjordingshandläggare, skulle kunna göra en sökning i systemet om en observatör ringde in för att rapportera att han sett en alien. Observatören skulle troligtvis uppge ett eller flera kännetecken på den alien han sett och telefonisten behövde därför kunna mata in dessa kännetecken och på så vis få fram vilken alien det rörde sig om och dess farlighet. Detta var väldigt viktigt för att en snabb åtgärd skulle kunna sättas in om en alien klassades högt på farlighetsskalan.

2.2.2 Fältagenten

En fältagent skulle, precis som telefonisten (se 2.2.1), kunna plocka fram farligheten på en alien via dess kännetecken. Detta för att han skulle veta när han fått syn på en alien om denna var klassad som farlig och förstärkning i så fall behövde inkallas.

En fältagent skulle även kunna plocka fram all övrig information om en specifik alien via dess ID-nummer om den var oregistrerad eller via dess personnummer om den var registrerad. Detta eftersom att om fältagenten gav sig ut på ett uppdrag så kunde det vara bra att plocka fram all information om de aliens som man redan hade identifierat och därmed förväntade sig att möta.

Fältagenten skulle även kunna plocka fram information om alla de aliens som han kunde förvänta sig att möta när han var på väg ut på ett uppdrag där man identifierat ett skepp och kunde uppge dess id i sökningen, men även om man bara kunde mata in dess kännetecken.

Till sist skulle en fältagent även kunna mata in nya aliens, raser och skepp och kunna skapa kopplingar mellan aliens, skepp och vapen allt eftersom han stötte på dem på sina uppdrag.

2.2.3 Administratören

En administratör skulle kunna mata in nya planeter som upptäcks och han skulle även kunna hemligstämpla en ras om detta efterfrågades.

2.3 Generell plan för applikationen

Den generella planen för applikationen var att användaren först skulle komma till en startsida där denne sedan kunde välja att logga in som telefonist, agent eller administratör. Beroende på användarens val skulle sedan de funktioner som denne var berättigad till bli synliga.

Om användaren valt att logga in som telefonist skulle enbart möjligheten att söka efter en alien via dess kännetecken bli synligt. Om användaren istället valt att logga in som agent eller administratör skulle istället deras respektive berättigade verktyg bli synliga enligt beskrivning i 2.2.

3 Databasen

Eftersom databasen skulle vara en prototyp så implementerades enbart det som tillhörde den avgränsning som gjorts. Ett antal tabeller skapades följt av triggers, vyer index och procedurer. I detta kapitel diskuteras kort hur koden ser ut samt tanken bakom.

3.1 Tabeller och optimering

Ett krav i uppgiftsbeskrivningen var att databasens tabeller skulle optimeras så mycket som bara var möjligt. Detta kapitel beskriver hur detta åstadkoms för den avgränsning som gjordes och nedanstående kod, som är ett urplock ur databasen, representerar databasens tabeller och dess olika optimeringar på ett bra sätt.

```
1 CREATE TABLE planet(  
2     nr TINYINT AUTO_INCREMENT,  
3     namn VARCHAR(20) UNIQUE NOT NULL,  
4     PRIMARY KEY(nr)  
5 ) ENGINE=INNODB;  
6  
7 CREATE TABLE skepp(  
8     skeppID CHAR(4) CHECK(skeppID LIKE '[0-9][0-9][0-9][0-9]'),  
9     sittPlatser SMALLINT DEFAULT '1' CHECK(sittPlatser>=1 AND  
10        sittPlatser<=5000),  
11     tillverkningsPlanet TINYINT NOT NULL,  
12     PRIMARY KEY(skeppID),  
13     FOREIGN KEY(tillverkningsPlanet) REFERENCES planet(nr)  
14 ) ENGINE=INNODB;
```

3.1.1 Datatypsoptimering

De olika datatyperna i tabellerna valdes noggrant ut utifrån vad de skulle innehålla för att få ett så optimalt system som möjligt. Exempelvis där datan förväntades vara ett tal som inte överskred värdet 127, som på rad 2 ovan, så valdes **tinyint**, och där textsträngen alltid skulle vara ett visst antal tecken lång valdes **char** och sedan specificerades antal tecken, som på rad 8 ovan. Även **smallint** användes på ett par ställen där datan förväntades kunna bli större än 127 men aldrig skulle överstiga 32767, som på rad 9 ovan där man tack vare uppgiftsbeskrivningen vet att max antal platser i skeppet är 5000. Även **varchar** användes på de ställen där antalet tecken som kunde matas in riskerade att variera, som på rad 3 ovan.

3.1.2 Restriktionsoptimering

Alla tabeller fick naturligtvis primärnycklar. Dessutom fick kodtabellerna (se 3.1.4) även attribut satta till **unique** på grund av att de skulle innehålla unika värden, exempelvis planeter som i tabellen planet ovan (se rad 3), som sedan kopplades samman med ett nummer. Namnet på planeten skulle alltså vara det unika men sedan skulle detta kopplas samman med ett nummer som skulle bli radens nyckel.

Ett flertal tabeller fick en eller flera **check**-satser som skulle säkerställa att reglerna som satts upp i uppgiftsbeskrivningen efterföljdes. Ex antalet sittplatser i ett skepp, se rad 9-10, där checken alltså kontrollerade så att antalet var minst 1 och max 5000. Kommandot check var dock inte aktivt i den SQL-version som databasen kördes i och därför fungerade istället dessa checkar mer som en påminnelse på vilka triggerar (se 3.4) som behövde skapas.

De flesta tabellerna fick ett eller flera attribut satta till **not null**. Detta för att försäkra sig om att all information som krävdes i raden vid en inmatning verkligen kom med, även om attributet inte var satt till **primary key** eller **unique** som automatiskt kräver en inmatning. Exempel ses på rad 3 ovan där en rad i tabellen planet naturligtvis inte var komplett utan ett

namn på den nya planeten som matats in. Detta innebar att både attribut som krävdes enligt uppgiftsbeskrivningen och attribut som krävdes för tabellens funktion alltså blev satta till not null.

3.1.3 Denormaliseringsoptimering

Tabellen "vapenbutiker" bröts isär i två, delvis eftersom den information som troligtvis skulle vara mest intressant att få fram vid en förfrågan på den var vilken butik som ägde vilket vapen, inte vart butiken låg, och delvis för att slippa upprepa både butiksnamn och adress för varje vapen som de sålde. Därav skapades istället två tabeller där den ena innehöll vilka vapen som hörde till vilken butik och den andra innehöll vilken butik det var, dvs. namn och adress. Se kod nedan.

```
15 CREATE TABLE butiker(  
16     nr TINYINT AUTO_INCREMENT,  
17     namn VARCHAR(20),  
18     planet TINYINT,  
19     PRIMARY KEY(nr),  
20     FOREIGN KEY(planet) REFERENCES planet(nr)  
21 ) ENGINE=INNODB;  
22  
23 CREATE TABLE vapenButiker(  
24     vapenID CHAR(5),  
25     butik TINYINT,  
26     PRIMARY KEY(vapenID,butik),  
27     FOREIGN KEY(vapenID) REFERENCES vapen(vapenID),  
28     FOREIGN KEY(butik) REFERENCES butiker(nr)  
29 ) ENGINE=INNODB;
```

3.1.4 Kod-tabeller

Ett flertal tabeller delades upp till kod-tabeller för att undvika redundans. De som blev uppdelade på detta vis var bland annat tabeller som innehöll planetnamn på varje rad. Istället för att upprepa en lång textsträng så behövdes endast ett tal som tog betydligt mindre plats finnas med i tabellen och kunde sedan länka till en annan tabell där det unika namnet på varje planet enbart fanns en enda gång (se rad 1-5 ovan). Även för ras och farlighet skapades varsin kodtabell.

3.1.5 Automatiserade värden

Ett antal tabeller fick automatiserade värden som alltså automatiskt matades in om inget annat angavs. Som exempel så matades värdet "1" in automatiskt tack vare kommandot **default** på rad 9 ovan om användaren inte angav antalet sittplatser i skeppet vid en ny inmatning. Detta på grund av detta krävdes i uppgiftsbeskrivningen. På samma sätt matades värdet "neutral" in om inget värde för farlighet angavs för en alien.

Även kommandot **auto_increment** användes för att automatiskt mata in nästkommande nummer i en serie på alla kodtabeller där alla värden kopplades samman med ett nummer. Tyvärr fick dock auto_increment på tabellen "ras" tas bort eftersom det annars orsakade att en procedur vägrade att fungera.

3.2 Vyer

Ett annat krav i uppgiftsbeskrivningen var att databasen skulle innehålla vyer så därför skapades tre stycken. En vy som sammanställer all information om aliens i en och samma imaginära tabell (se rad 30-39 nedan), en annan som sammanställer allt om ett skepp (se rad 41-46 nedan) och en tredje som sammanställer allt om ett vapen (se rad 48-55 nedan). Valet att sammanställa dessa tre gjordes för att misstanken fanns att användaren ofta skulle vilja plocka fram all information som fanns om en specifik alien, ett specifikt vapen eller ett specifikt skepp. För att underlätta detta så skapades alltså vyer så att man skulle slippa att ställa komplicerade frågeställningar där man kombinerade många tabeller.

```
30 CREATE VIEW alltOmEnAlien AS  
31 SELECT alien.alienID,oRegAlien.idKod,regAlien.pNr,alien.namn,ras.namn AS  
32     Ras, alienKannetecken.tecken,planet.namn AS Planet,farlighet.typ
```

```

33 FROM alien,farlighet,ras,oRegAlien,regAlien,planet,alienKannetecken
34 WHERE alien.farlighet=farlighet.nr AND
35     alien.ras=ras.nr AND
36     alien.alienID=oRegAlien.alienID AND
37     alien.alienID=regAlien.alienID AND
38     regAlien.planet=planet.nr AND
39     alien.alienID=alienKannetecken.alienID;
40
41 CREATE VIEW alltOmEttSkepp AS
42 SELECT skepp.skeppID AS ID,planet.namn AS TillverkningsPlanet,
43     skeppKannetecken.tecken AS Kannetecken
44 FROM skepp,planet,skeppKannetecken
45 WHERE skepp.tillverkningsPlanet=planet.nr AND
46     skepp.skeppID=skeppKannetecken.skeppID;
47
48 CREATE VIEW alltOmEttVapen AS
49 SELECT vapen.vapenID,planet.namn AS planetnamn,butiker.namn,farlighet.typ
50 FROM vapen,planet,farlighet,butiker,vapenButiker
51 WHERE vapen.tillverkningsPlanet=planet.nr AND
52     vapen.farlighet=farlighet.nr AND
53     vapen.vapenID=vapenButiker.vapenID AND
54     vapenButiker.butik=butiker.nr
55 ORDER BY vapenID ASC;

```

3.3 Index

Index behövde skapas för att optimera sökningar på data i databasen som inte var primärnycklar (primärnycklar indexeras automatiskt) men som användaren ändå ofta ville ställa frågor på. Då det antogs att både telefonisten och agenten skulle ha nytta av att kunna sortera aliens efter farlighet så skapades ett index över detta, se rad 56-57. Att sortera alla raser i bokstavsordning var också något som ansågs nödvändigt och därför skapades det ett index även över detta, se rad 59-60.

```

56 -- INDEX FOR QUERY SELECT * FROM alien ORDER BY farlighet;
57 CREATE INDEX farlighetPaAlien ON alien(farlighet DESC) USING BTREE;
58
59 -- INDEX FOR QUERY SELECT * FROM ras ORDER BY namn;
60 CREATE INDEX alienRaser ON ras(namn ASC) USING BTREE;

```

3.4 Triggers

Ett fåtal triggers skapades för att komplettera det check'ar (se 3.1.2) i tabellerna som ju inte var funktionella i denna version av SQL. En trigger förhindrade därmed att man kunde mata in fler än 15 kopplingar mellan en alien och ett vapen eftersom en alien enligt uppgiftsbeskrivningen inte fick äga mer än just 15 vapen (se rad 61-69). Om användaren ändå försöker med detta försöker triggern att göra en omöjlig uppdatering i en tabell vilket resulterar i att ett felmeddelande skrivs ut i form av det som angivits i triggern. Detta beror på att en korrekt felhantering idag inte stöds i det system som databasen ligger på. Ytterligare två triggers skapades på liknande sätt, varav en såg till att man inte kunde göra mer än en koppling från ett unikt regnr till ett oregnr eftersom en alien enbart borde kunna registreras en gång i vardera register, och en annan trigger såg till att ID-nummer på ett skepp måste bestå av just 4 siffror eftersom det kändes som en lämplig form på ett ID-nummer för skepp.

```

61 DELIMITER //
62 CREATE TRIGGER vapenAgarGrans BEFORE INSERT ON vapenAgarAlien
63 FOR EACH ROW BEGIN
64     IF((SELECT COUNT(*) FROM vapenAgarAlien WHERE alien=NEW.alien)=15)
65     THEN
66         UPDATE vapen SET En_alien_kan_inte_aga_mer_an_femton_vapen=0;
67     END IF;
68 END;
69 // DELIMITER ;

```

3.5 Procedurer

Till sist skapades två procedurer i databasen som beskrivs i detta kapitel.

3.5.1 Hemligstämpla en ras

Eftersom det var ett krav i uppgiftsbeskrivningen att man skulle kunna hemligstämpla en ras så skapades proceduren "hemligstamplaRas" som citeras nedan. På rad 70-86 så deklareras först tre stycken tabeller som skall användas som loggtabeller, vilket innebär att all information som sedan raderas av proceduren kommer att sparas i dessa tabeller tillsammans med tidpunkten för att man sedan vid behov skall kunna återställa rasen. Inga nycklar anges eftersom man eventuellt vill kunna radera samma ras flera gånger och nycklar kräver unika värden.

På rad 91-93 startar proceduren först med att kontrollera om det finns en ras som heter "Hemligstämplad", gör det inte det så skapas en sådan i rastabellen. Därefter på rad 95-108, så sparas nödvändig information i loggtabellerna, och sedan, på rad 110-116, så byts rasen på alla refererande rader ut till Hemligstämplad. Till sist raderas rasen från rastabellen.

På rad 122 deklareras ett hårdkodat anrop till proceduren som använts för att testa dess funktion.

```
70      CREATE TABLE rasLogg(  
71          tid DATETIME,  
72          nr SMALLINT,  
73          namn VARCHAR(20)  
74      ) ENGINE=INNODB;  
75  
76      CREATE TABLE alienLogg(  
77          tid DATETIME,  
78          alienID CHAR(25),  
79          ras SMALLINT NOT NULL  
80      ) ENGINE=INNODB;  
81  
82      CREATE TABLE rasKanneteckenLogg(  
83          tid DATETIME,  
84          rasNamn SMALLINT,  
85          tecken VARCHAR(20)  
86      ) ENGINE=INNODB;  
87  
88      DELIMITER //  
89      CREATE PROCEDURE hemligstamplaRas(num SMALLINT)  
90      BEGIN  
91          IF(NOT EXISTS(SELECT * FROM ras WHERE namn='Hemligstämplad')) THEN  
92              INSERT INTO ras(nr,namn) VALUE('0','Hemligstämplad');  
93          END IF;  
94  
95          INSERT INTO rasLogg(tid,nr,namn)  
96          SELECT now(),nr,namn  
97          FROM ras  
98          WHERE ras.nr=num;  
99  
100         INSERT INTO alienLogg(tid,alienID,ras)  
101         SELECT now(),alienID,ras  
102         FROM alien  
103         WHERE alien.ras=num;  
104  
105         INSERT INTO rasKanneteckenLogg(tid,rasNamn,tecken)  
106         SELECT now(),rasNamn,tecken  
107         FROM rasKannetecken  
108         WHERE rasKannetecken.rasNamn=num;  
109  
110         UPDATE alien  
111         SET alien.ras=0  
112         WHERE alien.ras=num;  
113  
114         UPDATE rasKannetecken  
115         SET rasKannetecken.rasNamn=0  
116         WHERE rasKannetecken.rasNamn=num;  
117
```

```

118      DELETE FROM ras WHERE nr=num;
119  END;
120  // DELIMITER ;
121
122  -- CALL hemligstampLaRas(2);

```

3.5.2 Radera en alien

Ett annat krav i uppgiftsbeskrivningen var att man skulle kunna radera en alien och alla länkar till denna vid behov. Därav skapades ytterligare en procedur som gick att kalla på, se rad 123-144 nedan. Proceduren går igenom alla tabeller som kan tänkas vara sammankopplade med den alien som skall raderas och raderar kopplingarna till denna i tur och ordning. Om ett vapen enbart har denna alien som ägare raderas även vapnet och alla dess kopplingar eftersom detta var ytterligare ett krav i uppgiftsbeskrivningen.

För att kunna radera kopplingarna i rätt ordning måste man först radera kopplingen mellan vapen och alien innan man kan radera vapnet, men gör man detta så kan man ju inte sedan plocka fram vilka vapen som är sammankopplade till den aktuella alien. Lösningen på detta synes på rad 131-138 där proceduren utför ett mer avancerat steg och först skapar en temporär tabell och sedan sparar alla ID-nummer på de vapen som enbart den alien som skall raderas står som ägare på. Därefter kan man radera i de ordinarie tabellerna och till sist kasta bort den temporära tabellen.

```

123  DELIMITER //
124  CREATE PROCEDURE raderaAlien(id CHAR(25))
125  BEGIN
126      DELETE FROM alienKannetecken WHERE alienID=id;
127      DELETE FROM omReg WHERE oRegAlien=id;
128      DELETE FROM bekantskap WHERE alien1=id OR alien2=id;
129      DELETE FROM skeppsCrew WHERE alienID=id;
130
131      CREATE TABLE temp(nr CHAR(5));
132      INSERT INTO temp(nr) SELECT vapen FROM vapenAgarAlien WHERE alien=id
133          AND vapen NOT IN (SELECT vapen FROM vapenAgarAlien WHERE NOT
134              alien=id);
135      DELETE FROM vapenButiker WHERE vapenID IN (SELECT * FROM temp);
136      DELETE FROM vapenAgarAlien WHERE alien=id;
137      DELETE FROM vapen WHERE vapenID IN (SELECT * FROM temp);
138      DROP TABLE temp;
139
140      DELETE FROM regAlien WHERE alienID=id;
141      DELETE FROM oRegAlien WHERE alienID=id;
142      DELETE FROM alien WHERE alienID=id;
143  END;
144  // DELIMITER ;

```

4 PHP-applikationen

PHP-applikationen skulle precis som databasen vara en prototyp och behövde därmed inte stödja alla funktioner och möjligheter som databasen innehöll, dock fanns ett antal krav som beskrivs i detta kapitel i tur och ordning tillsammans med dess lösning.

De funktioner som implementerades var "Inmatning av ett nytt skepp" och "Sökning av specifik alien" som agenten antogs ha nytta av, samt "Hemligstämpling av ras" som databasadministratören antogs ha nytta av.

4.1 Uppdatering av databasen

Då det krävdes att det skulle finnas med en uppdatering av databasen valdes möjligheten att kunna mata in ett nytt skepp i systemet. Det första som krävdes för denna funktion var därmed ett formulär. På rad 1 nedan skapades ett formulär som hämtar in information från användaren och när denne trycker på skicka-knappen laddas sidan om och bifogar den information som användaren matat in. På rad 5-7 beskrivs ett textfält där användaren förväntas mata in en fyrsiffrig sifferkombination som därmed blir ID-numret på det nya skeppet.

```

1 <form action="skeppinsert.php" method="post">
2 <table border="0">
3 <tr><td>Skeppets ID:</td>
4 <td>
5 <input size="25" type="text" value="4 siffror" name="id"
6 onblur="if (this.value == '') {this.value = '4 siffror'};"
7 onfocus="if (this.value == '4 siffror') {this.value = ''};" />
8 </td></tr>

```

Eftersom användaren inte skulle behöva mata in en ny planet utan istället kunna välja på de befintliga i systemet löstes detta med nedanstående PHP-kod. På rad 10-12 skapas en option-box och på rad 13-17 körs en foreach-loop som itererar igenom alla planeter i databasen och stoppar in dessa som valbara alternativ i option-boxen.

```

9 <?php
10 echo '<select size="1" name="tillverkningsplanet">';
11 echo '<option value="" selected="selected">Välj en
12 planet'.nbsp(20).</option>';
13 foreach($pdo->query( 'SELECT namn FROM planet ORDER BY namn;' ) as $row){
14 echo '<option value="'. $row['namn']. '">';
15 echo $row['namn'];
16 echo '</option>';
17 }
18 echo '</select>';
19 ?>

```

På i princip samma sätt som i kodstycket ovan skapades ytterligare en option-box utifrån de kännetecken som aliens kunde ha men där man istället kunde välja flera alternativ. Tanken med denna var att man skulle kunna klicka i flera kännetecken som en alien hade på en gång. Denna funktion färdigställdes dock inte i brist på tid utan finns mest där som en påminnelse om vilka funktioner som skulle kunna implementeras i PHP-applikationen.

Resultatet på hur formuläret ser ut i webbläsaren ses på Bild 2 nedan.

Bild 2 Inmatning av nytt skepp via PHP-applikationen.

När väl användaren laddar om sidan via skicka-knappen och därmed förhoppningsvis bifogar inmatad data så hanteras detta av ytterligare ett block med PHP-kod som bitvis citeras nedan.

Först så kontrolleras den inmatade datan med ett antal if-satser som kontrollerar om datan alls har blivit ifylld, om det gjorts i korrekt form och så vidare. På rad 20 nedan kontrolleras exempelvis om ID-numret har matats in som 4 siffror.

```

20 if ctype_digit ($id) and strlen($id)==4){

```

Rad 21 kontrollerar istället om antalet sittplatser matats in som ett heltal mellan 1 och 5000.

```
21     else if($_POST['sittplatser']>=1 || $_POST['sittplatser']>=5000){
```

Om allt är i sin ordning så sparas datan i variabler som sedan binds samman med parametrar i en querystring, se rad 22-28. Därefter körs frågeställningen mot databasen, se rad 29.

```
22     $querystring='INSERT INTO skepp(skeppID,sittPlatser,tillverkningsPlanet)
23     VALUES (:SKEPPID, :SITTPLATSER, :PLANET)';
24
25     $stmt = $pdo->prepare($querystring);
26     $stmt->bindParam(':SKEPPID', $id);
27     $stmt->bindParam(':SITTPLATSER', $sittplatser);
28     $stmt->bindParam(':PLANET', $planetnr);
29     $stmt->execute();
```

Till sist skrivs en tabell ut på skärmen som med hjälp av en foreach-loop (se rad 30-41) går igenom alla rader i tabellen "skepp" och matar ut dessa, rad för rad. En inre foreach-loop (se rad 36-38) översätter dessutom koden för vilken planet skeppet tillhör, till dess namn i tabellen.

```
30     foreach($pdo->query( "SELECT * FROM skepp WHERE skeppID='$id';" ) as $row){
31         echo '<tr>';
32         echo '<td>'.$row['skeppID'].'</td>';
33         echo '<td>'.$row['sittPlatser'].'</td>';
34         $p=$row['tillverkningsPlanet'];
35         echo '<td>';
36         foreach($pdo->query( "SELECT namn FROM planet WHERE nr='$p';" ) as $row){
37             echo $row['namn'];
38         }
39         echo '</td>';
40         echo '</tr>';
41     }
```

Resultatet i webbläsaren blir som exempel enligt Bild 3 nedan.

Följande information har registrerats i databasen:

SkeppsID	Antal sittplatser	Tillverkningsplanet
1567	3450	Saturnus

Bild 3 Resultat efter ny inmatning av skepp i PHP-applikationen.

4.2 Exekvering av en procedur

Att kunna exekvera en procedur ifrån PHP-applikationen var även det ett krav och valet hamnade på "Hemligstämpla en ras" (se 3.5.1).

Lösningen var enkel och gjordes i form av ett formulär med en option-box som precis som i föregående exempel (se rad 9-19 ovan) itererar igenom alla raser som finns i tabellen ras med hjälp av en foreach-loop och stoppar in dessa som valbara alternativ i option-boxen. När användaren klickar på skicka-knappen så laddas sidan om och den ras som valts finns då inte längre med i option-boxen. Exempel på hur det kan se ut i webbläsaren syns på Bild 4.

Välj ras att hemligstämpla

Välj en ras ▼ Hemligstämpla

- Välj en ras
- Avatar
- Cybertron
- Gremlin
- Reptoid
- Transformer
- Velociraptor

Bild 4 Hemligstämpling av ras i PHP-applikationen.

När sidan laddas om så kontrollerar ett kod-block om en ras valts och sparar detta i en variabel (se rad 42). Denna variabel skickas sedan som en fråga till databasen (se rad 43-45) för att få fram det nr som representerar rasen som valt eftersom proceduren kräver kodnummer på rasen och inte dess namn för att kunna köras.

```

42     $namn=$_POST['ras'];
43     foreach($pdo->query( "SELECT nr FROM ras WHERE namn='$namn';" ) as $row){
44         $nr=$row['nr'];
45     }

```

Till sist kallas proceduren och bifogar den variabel som tagits fram från ovanstående kodblock som parameter, och därefter laddas sidan återigen om för att användaren skall kunna se korrekt innehåll i option-boxen.

```

46     $pdo->query( "CALL hemligstamplaRas('$nr');" );

```

Resultatet av det kan se ut exempelvis som på Bild 5 där rasen Transformer valts att hemligstämplas, jämför detta med Bild 4 Hemligstämpling av ras i PHP-applikationen..

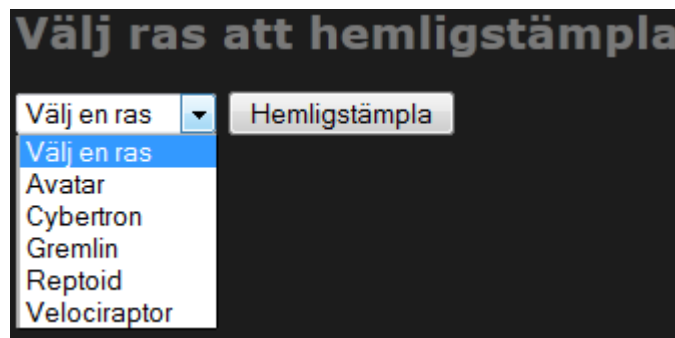


Bild 5 Resultat efter hemligstämpling av ras i PHP-applikationen

Rasen "Hemligstämplad" som skapas när man hemligstämplar en ras (om den inte redan finns) visas aldrig i option-boxen eftersom användaren ju inte ska kunna ta bort denna.

4.3 Sökning

Att kunna söka på något i databasen var också något som användaren behövde kunna göra via PHP-applikationen. Valet föll på att söka efter en alien med hjälp av dess ID-kod eller personnummer.

Först skapades ett formulär där användaren kunde mata in numret i en text-box. Även tre stycken option-boxar itererades fram med hjälp av en do-loop i PHP (se rad 48-56) och vardera option-box fylldes med de kännetecken som en alien kan ha. Detta gjordes med hjälp av en inre foreach-loop (se rad 51-56) som itererade fram alla kännetecken i tabellen "alienKannetecken".

```

47     $i=1;
48     do{
49         echo '&nbsp;<select size="1" name="sign'.$i++.'">';
50         echo '<option value="" selected="selected">Välj ett kännetecken</option>';
51         foreach($pdo->query( 'SELECT tecken FROM alienKannetecken ORDER BY tecken;' )
52             as $row){
53             echo '<option value="'.$row['tecken'].'">';
54             echo $row['tecken'];
55             echo '</option>';
56         }
57         echo '</select>';
58     }while($i<=3)

```

På Bild 6 ses ett exempel på hur sökformuläret ser ut i webbläsaren. Känneteckens-boxarna har dock i nuläget ingen funktion då detta inte implementerats i brist på tid men tanken var att användaren skulle kunna välja ett eller flera kännetecken och på så vis hitta alla de aliens med dessa egenskaper i kombination.

Sök efter Alien

Mata in idnr(12 siffror), pnr(10 siffror) eller välj ett eller flera kännetecken att söka på:

- Brun
- Orange

Bild 6 Sökning efter alien i PHP-applikationen

När användaren trycker på sök-knappen laddas sidan om och ett block med PHP-kod styr vad som händer sedan. Det som först händer är att användarens input kontrolleras med hjälp av if-satser. Om den består av 10 eller 12 siffror (se rad 59) så stoppas ett bindestreck in mellan siffrorna (se rad 60) och därefter skickas frågan till databasen och resultatet skrivs ut i en tabell med hjälp av en foreach-loop (se rad 62-75).

```

59 if ctype_digit ($nr) and strlen($nr)==12){
60     $idnr=implode('-', str_split($nr,6));
61
62     echo '<table border="1">';
63     foreach($pdo->query( "SELECT * FROM alltOmEnAlien WHERE pNr='$pnr';" ) as $row){
64         echo '<tr>';
65         echo '<td>'.$row['namn'].'</td>';
66         echo '<td>'.$row['alienID'].'</td>';
67         echo '<td>'.$row['idKod'].'</td>';
68         echo '<td>'.$row['pNr'].'</td>';
69         echo '<td>'.$row['Ras'].'</td>';
70         echo '<td>'.$row['tecken'].'</td>';
71         echo '<td>'.$row['Planet'].'</td>';
72         echo '<td>'.$row['typ'].'</td>';
73         echo '</tr>';
74     }
75     echo '<table>';

```

Om inputen däremot inte är korrekt så skrivs information om detta ut på skärmen. Exempel på resultatet från en korrekt input ses på Bild 7 nedan.

ID:	4564532956326565652989721
ID-kod:	120924-121212
P-nr:	120925-1564
Namn:	Kiwi
Ras:	Hemligstämplad
Kännetecken:	Brun
Ursprungsplanet:	Saturnus
Risk:	Harmlös

Bild 7 Resultatet efter en sökning efter alien i PHP-applikationen

4.4 Kontrasten mellan PDO i PHP och formulär i .NET

Den största kontrasten mellan PHP och .NET är att medan .NET är händelsebaserat och kallar på funktioner, dvs objektorienterat, så är PHP istället mer procedurellt, dvs att det som händer när användaren trycker på knappen är att sidan laddas om och utifrån den input som användaren givit kan nya objekt göras synliga på webbsidan.

Vid arbete mot databasen så känns PDO'n i PHP mer rättfram eftersom man ställer alla frågeställningar mot databasen med ett och samma verktyg, medans man i .NET måste använda sig utav flera olika funktioner för att kunna ställa olika typer av frågor till databasen.

5 .NET-applikationen

Liksom i PHP så behövde .NET-applikationen bara implementera en del utav databasen men den måste ändå uppfylla ett antal krav som fanns beskrivna i uppgiftsbeskrivningen. I detta kapitel diskuteras dessa krav tillsammans med sina lösningar i tur och ordning.

De funktioner som implementerades var "Radera alien" och "Sök efter vapen" som båda förväntades användas utav en agent, samt "Mata in ny planet" som förväntades användas av databasadministratören.

5.1 Paneler

.NET-applikationen byggdes upp med paneler i ett och samma dokument och dessa styrdes sedan av funktioner som talade om vilka paneler som för tillfället skulle vara synliga.

5.1.1 Välkomstpanelen

Den första panelen som skapades var välkomstpanelen (se rad 1-14). Det enda den innehåller är koden för att lite text och fyra stycken knappar ska visas i webbläsaren. Varje knapp är deklarerad till att när användaren trycker på knappen så kallar den på en funktion som byter ut vilka paneler som ska vara synliga på sidan.

```
1 <asp:Panel id="welcomePanel" runat="server">
2 <h1>V&auml;kommen till .NET-sidan!</h1>
3 <p>Vad vill du g&ouml;ra?</p>
4 <form id="toDoForm" runat="server">
5 <asp:Button ID="toDoAlien" runat="server" onclick="alienFormFunktion"
6 Text="Radera alien" />&nbsp;
7 <asp:Button ID="toDoPlanet" runat="server" onclick="planetFormFunktion"
8 Text="Mata in ny planet" />&nbsp;
9 <asp:Button ID="toDoVapen" runat="server" onclick="vapenFormFunktion"
10 Text="S&ouml;ka efter vapen" />&nbsp;
11 <asp:Button ID="RefreshButton" runat="server" onclick="RefreshCode"
12 Text="Ladda om sidan" /><br/>
13 </form>
14 </asp:Panel>
```

Välkomstpanelen ser ut som på Bild 8 i webbläsaren och ligger alltid som synlig på sidan.



Bild 8 Välkomstpanelen i .NET-applikationen

5.1.2 Synliggöra och dölja paneler

Om användaren väljer att klicka på exempelvis knappen "Radera alien" (se Bild 8) så kommer en funktion att köras som har som enda uppgift att låta välkomstpanelen fortfarande vara synlig (se rad 16), göra panelen med formuläret för att radera en alien synlig (se rad 17) och se till att alla andra paneler är dolda (se rad 18-22).

```
15 private void alienFormFunktion(Object sender, EventArgs e){
```

```

16     welcomePanel.Visible=true;
17     alienFormPanel.Visible=true;
18     alienPanel.Visible=false;
19     planetFormPanel.Visible=false;
20     planetPanel.Visible=false;
21     vapenFormPanel.Visible=false;
22     vapenPanel.Visible=false;
23 }

```

På samma sätt finns en i princip likadan funktion även för knapparna "Mata in ny planet" och "Söka efter vapen". Den sista knappen "Ladda om sidan" kallar dock på en helt annan typ av funktion som används för att "refresha" sidan medan den byggs.

5.2 Exekvering av en procedur

Ett krav i .NET-applikationen var att användaren skulle kunna kalla på en procedur i databasen och detta valdes att göras med funktionen "Radera alien" (se 3.5.2).

5.2.1 Formuläret

När användaren tryckt på knappen "Radera alien" (se Bild 8) och den tillhörande funktionen körts så blir alltså ytterligare en panel synlig som innehåller ett formulär med en text-box som frågar efter ett ID-nummer (se rad 24), samt en knapp som sedan kallar på den funktion som används för att radera en alien ur databasen (se rad 25-26).

```

24     <asp:TextBox ID="alienid" runat="server"/>
25     <asp:Button ID="alienButton" runat="server" onclick="DeleteAlien"
26     Text="Radera" />

```

I webbläsaren ser panelen ut enligt Bild 9 nedan.

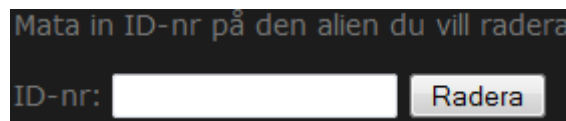


Bild 9 Formuläret för att radera en alien i .NET-applikationen

5.2.2 Funktionen

Funktionen som knappen kallar på öppnar först upp en kanal mellan .NET-applikationen och databasen (se rad 27-31).

```

27     string connectionString = "Server=localhost;Database=allemmjo;User
28     ID=dbsk;Password=Tomten2009;Pooling=false;Character Set=latin1;";
29
30     MySqlConnection dbcon = new MySqlConnection(connectionString);
31     dbcon.Open();

```

Därefter skapas en kommandosträng och användarens input binds till en variabel som skickas med som parameter vid anropet av proceduren (se rad 32-37).

```

32     string strInsert="CALL raderaAlien(@alienid);";
33     MySqlCommand sqlCmd=new MySqlCommand(strInsert, dbcon);
34
35     sqlCmd.Parameters.Add("@alienid", alienid.Text);
36
37     sqlCmd.ExecuteNonQuery();

```

Som avslutning innan kanalen stängs ner så skickas ytterligare en förfrågan till databasen efter information om alla de aliens som finns kvar i databasen efter raderingen (se rad 38-39). Resultatet sparas i en tabell som ritas ut i ytterligare en panel som funktionen avslutar med att göra synlig (se rad 40-53).

```

38     MySqlDataAdapter adapter = new MySqlDataAdapter("SELECT * FROM alltOmEnAlien",
39     dbcon);
40     DataSet ds = new DataSet();
41     adapter.Fill(ds, "result");
42
43     alienCustomerGrid.DataSource = ds.Tables["result"];

```



```

44     alienCustomerGrid.DataBind();
45     dbcon.Close();
46
47     welcomePanel.Visible=true;
48     alienFormPanel.Visible=true;
49     alienPanel.Visible=true;
50     planetFormPanel.Visible=false;
51     planetPanel.Visible=false;
52     vapenFormPanel.Visible=false;
53     vapenPanel.Visible=false;

```

5.2.3 Resultatspanelen

Panelen som visar upp resultatet och som blir synlig först efter att funktionen körts innehåller i princip enbart ett kommando som ritar ut den tabell som funktionen skapat (se rad 54-55). Dock så innehåller den även information om hur tabellens kolumnrubriker skall se ut (se rad 56-59) eftersom att detta var tvunget då problem med att visa upp åäö annars uppstod.

```

54     <asp:DataGrid ShowHeader="true" AutoGenerateColumns="false" runat="server"
55     id="alienCustomerGrid">
56     <HeaderStyle CssClass="header"></HeaderStyle>
57     <Columns>
58     <asp:BoundColumn DataField="idKod" HeaderText="<div id='headfont'>Kod</div>"
59     ReadOnly="True" />

```

När panelen skrivs ut i webbläsaren så ser den exempelvis ut som på Bild 10 där två återstående aliens finns kvar i databasen. Valet att utelämnas kolumnen som innehåller ID-numret gjordes av praktiska skäl för att tabellen skulle få plats i det gränssnitt som skapats på webbplatsen.

Följande aliens finns kvar i registret:

Kod	Pnr	Namn	Ras	Kännetecken	Planet	Farlighet
120924-255255	120925-9873	Mango	Hemligstämplad	Orange	Mars	Halvt harmlös
120924-121212	120925-1564	Kiwi	Hemligstämplad	Brun	Saturnus	Harmlös

Bild 10 Resultatspanelen efter en radering av en alien i .NET-applikationen

5.3 Uppdatering av databasen

Ett annat krav i .NET-applikationen var att användaren skulle kunna uppdatera databasen på något sätt och valet blev att kunna mata in en ny planet.

5.3.1 Formuläret

Precis som i exemplet med "Radera alien" (se 5.2) så synliggörs en formulärspanel när användaren klickar på "Mata in ny planet"-knappen (se Bild 8). Koden för panelen är i princip identisk med rad 24-26 i kapitlet 5.2.1 och resultatet i webbläsaren ser ut enligt Bild 11 nedan.

Mata in ny planet

Namn:

Bild 11 Formuläret för att mata in en ny planet i .NET-applikationen

5.3.2 Funktionen

Funktionen för att mata in en ny planet är även den mycket lik den som beskrivs i kapitlet 5.2.2. Det som skiljer är att istället för en sträng där en procedur kallas, så skickas en sträng där en inmatning görs i databasen. Användarens input bifogas som parameter och resultatet skrivs sedan ut i en tabell i en ny panel som synliggörs innan funktionen avslutas.

5.3.3 Resultatspanelen

Precis som det beskrivs i kapitel 5.2.3 så innehåller även denna panel resultatet i form av en tabell där tabellens kolumnrubriker modifierats för att åäö ska fungera. I webbläsaren ser det exempelvis ut som på Bild 12.

Följande planeter finns nu i registret:

Nr	Namn
2	Mars
4	Merkurius
3	Neptunus
1	Saturnus

Bild 12 Resultatspanelen efter en inmatning av en ny planet i .NET-applikationen

5.4 Sökning

Det sista kravet som skulle uppfyllas i .NET-applikationen var att användaren skulle kunna göra en sökning i databasen. Valet föll på att kunna söka på all tillgänglig information om ett specificerat vapen.

5.4.1 Formuläret

Precis som i exemplet med "Radera alien" (se 5.2) så synliggörs en formulärspanel när användaren klickar på "Söka efter vapen"-knappen (se Bild 8). Koden för panelen är i princip identisk med rad 24-26 i kapitlet 5.2.1 och resultatet i webbläsaren ser ut enligt Bild 13 nedan.

Mata in ID-nr på det vapen du vill söka information om

ID-nr:

Bild 13 Formuläret för att söka efter ett vapen i .NET-applikationen

5.4.2 Funktionen

Funktionen för att söka i databasen påminner väldigt mycket om funktionen för att radera en alien och för att mata in en ny planet. Det som skiljer är att man inte först har ett stycke kod som används för att modifiera databasen utan man enbart har med den kod som hämtar data ur databasen, se rad 38-41 i kapitel 5.2.2. Dock petar man in en extra rad för att kunna skicka med användarens input som parameter till frågeställningen (se rad 60 nedan).

```
60 adapter.SelectCommand.Parameters.Add("@vapenid", vapenid.Text);
```

5.4.3 Resultatspanelen

Precis som det beskrivs i kapitel 5.2.3 så innehåller även denna panel resultatet i form av en tabell där tabellens kolumnrubriker modifierats för att åäö ska fungera. I webbläsaren ser det exempelvis ut som på Bild 14.

Detta är all information som gick att hitta om det valda vapnet i registret:

ID	Produceras på	Inköpsbutik	Farlighet
00001	Mars	Rhondas bang-bang	Halvt harmlös

Bild 14 Resultatspanelen efter en sökning på vapen i .NET-applikationen

5.5 Kontrasten mellan PDO i PHP och formulär i .NET

Se kommentarer på detta i kapitel 4.4.

5.6 Att arbeta med ADO

Det känns aningen omständigt att arbeta med ADO, dvs adapters, datasets osv. Det krävs uppenbarligen många funktioner för att kunna utföra olika saker när man använder .NET som API för en databas vilket inte upplevs som helt tillfredsställande.

6 Reflektioner

I detta kapitel diskuteras arbetet som sådant och andra reflektioner runt det.

6.1 Tankar om mitt arbete

Detta arbete har varit väldigt tungt av flera olika skäl. Bland annat så har det varit jobbigt att sitta och implementera databasen utan att veta vilka behov som sedan funnits för PHP-applikationen respektive .NET-applikationen. Att inte veta vilka möjligheter som finns i vardera API har också varit väldigt frustrerande och jag har stött på en hel del efter arbetets gång som jag har fått ändra och som jag velat ändra men inte hunnit.

Bland annat så hade jag velat göra fler triggers och procedurer. Förslag på dessa står dock med som kommentarer i källkoden. Sedan har jag reagerat på att jag borde ha använt mig utav `auto_increment` på skeppets ID och jag borde ha haft med vilka som äger ett visst vapen i `alltOmEttVapen`. Jag borde även ha sett till att binda alla parametrar som skickades via PHP-applikationen till databasen för att göra det säkrare, (som det ser ut nu så är mitt system väldigt osäkert), haft en `if-sats` som sa ifrån om användaren försökte radera en alien som inte existerade, haft felhantering i största allmänhet i min .NET-applikation med `if-satser` och liknande och jag borde inte ha skrivit ut kodnumren för tabellerna när planettabellen skrevs ut efter att man matat in en ny planet i databasen. Det är med andra ord väldigt mycket jag skulle ha ändrat på om jag fått chansen att göra om denna uppgift.

Det som jag ändå har varit relativt nöjd med är hur det slutgiltiga upplägget av .NET applikationen ser ut med alla sina paneler.

6.2 Tankar om kursen

Angående kursen så tycker jag att det kursmaterial som funnits har varit bra men det har varit irriterande med en del felstavningar och att demonstrationerna av exempelkoderna inte har fungerat. Jag tycker också att det har varit försvinnande lite information om .NET'en överhuvudtaget och jag hade gärna sett fler exempel som visade på fler möjligheter och dessutom hade jag föredragit en kort föreläsning med några tips och eventuellt lite inspiration på varje .NET-handledning.

Angående mjukvaran så tycker jag att Squirrel inte är ett vidare bra program då man inte ens kan ångra en feltryckning eller liknande med `Ctrl+Z`. Efter en del efterforskningar så installerades istället MySQL Workbench på min egen dator och detta program har fungerat ypperligt.

Till sist, om det stämmer att den version av .NET som ingått i denna kurs är förlegad så tycker jag verkligen att man istället skulle ha valt att använda sig av en nyare version, alternativt arbetat mer intensivt med enbart PHP och SQL.

6.3 Interaktiva tekniker

Då jag inte har jobbat vare sig med JavaScript, AJAX eller Flash tidigare så vet jag inte vilka möjligheter som finns med dessa verktyg men jag tror absolut att sidan hade kunnat bli betydligt trevligare om man hade kunnat integrera dessa.