

DYNAMIC MULTIMEDIA DATABASES

Chris Howarth, 0407805

General Information	
Course	MMTD
Level	3
Module Code	EE3103
Module Name	Databases for Multimedia Applications
Assignment Code	EE3103/1
Actual Date of Submission	19/01/2007
Title of Assessment	Dynamic Multimedia Databases
Set By	Dr. Paul Kyberd
Nature of Submission	Part 1 Submission
Author	Christopher Howarth
Student Registration Number	0407805
Email Address	ee04cjh@brunel.ac.uk
Abstract	
<p>This document reports on the development and implementation of a Web database application. PHP scripts accessing a mySQL database generate dynamic Web pages.</p>	
Keyword List	
Database Multimedia Dynamic	
Contents	
<p>1.1 – Use Cases 1.2 – Entities and their Attributes 1.3 – Tables and their Attributes 1.4 – Normalization of Tables 1.5 – The External Model 1.6 – The Internal Model</p> <p>3.1 – SQL Commands 3.2 – System Description 3.3 – Implementation Description 3.4 – Indexing 3.5 – Denormalisation</p>	

Table Design and Documentation

1.1 Use Cases

1. Search Website for Flight

Steps:

1. Customer selects Flight option
2. Customer selects a Departure city
3. Customer selects departure date and time
4. Customer selects a Destination city
5. Customer selects return date and time
6. Customer selects number of seats required
7. Customer presses the search button to search for available flights

Entities:

Customer
Flight

2. Search Website for Hotel

Steps:

1. Customer selects Hotel option
2. Customer selects a city
3. Customer selects Check-in date
4. Customer selects number of nights
5. Customer selects number of rooms required
6. Customer selects star rating
7. Customer presses the search button to search for available hotels

Entities

Customer
Hotel

3. Search Website for Holiday

Steps:

1. Customer selects Holiday option
2. Customer selects a Departure city
3. Customer selects departure date and time for the flight
4. Customer selects a Destination city
5. Customer selects return date and time for the flight
6. Customer selects number of seats required on the flight
7. Customer selects star rating for hotel
8. Customer selects number of rooms required in the hotel
9. Customer presses the search button to search for available flights

Entities

Customer
Flight
Hotel
Holiday

4. Register Details

Steps:

1. Customer requests the register page
2. Customer fills in the form entries
3. Customer submits the form to create account

Entities

Customer
Account

5. Change Registered Details

Steps:

1. Customer requests the account page
2. Customer updates the details
3. Customer submits the form

Entities

Customer
Account

6. Book Flight

Steps:

1. Customer must search for a flight (Use Case 1)
2. Customer must register details (Use Case 4)
3. Customer confirms details of flight are correct
4. Customer enters payment details
5. Customer confirms booking

Entities

Customer
Flight
Booking

7. Book Hotel

Steps:

1. Customer must search for a hotel (Use Case 2)
2. Customer must register details (Use Case 4)
3. Customer confirms details of hotel are correct
4. Customer enters payment details
5. Customer confirms booking

Entities

Customer
Hotel
Booking

8. Book Holiday

Steps:

1. Customer must search for a holiday (Use Case 3)
2. Customer must register details (Use Case 4)
3. Customer confirms details of hotel are correct
4. Customer confirms details of flight are correct
5. Customer enters payment details
6. Customer confirms booking

Entities

Customer
Hotel
Flight
Booking
Holiday

9. Employee Books Flight

Steps:

1. Employee searches for flight details (Use Case 1)
2. Employee takes customer details
3. Employee confirms booking to customer
4. Employee books flight

Entities

Employee
Customer
Flight
Booking

10. Employee Books Hotel

Steps:

1. Employee searches for hotel details (Use Case 2)
2. Employee takes customer details
3. Employee confirms booking to customer
4. Employee books hotel

Entities

Employee
Customer
Hotel
Booking

11. Employee Books Holiday

Steps:

1. Employee searches for holiday details (Use Case 3)
2. Employee takes customer details
3. Employee confirms flight and hotel booking to customer
4. Employee books holiday

Entities

Employee
Customer
Flight
Hotel
Holiday
Booking

1.2 Entities and their Attributes

1.21 Entity Frequency

Entity	Use Cases Occurred In
Customer	1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11
Flight	1, 3, 6, 8, 9, 11
Hotel	2, 3, 7, 8, 10, 11
Holiday	3, 8, 11
Account	4, 5
Booking	6, 7, 8, 9, 10, 11
Employee	9, 10, 11

1.22 Entity Attributes

Customer: Customer ID, Surname, Forename, Address, Phone, Mobile, Email, DOB, Info

**The 'info' attribute would contain a 'Y' or 'N' so as to say whether the customer would want to receive further updates*

Flight: Flight Code, Departure City, Destination City, Flight Time, Airline, Airline Code, Airline Address, Airline Phone, Seats Available, Cost

Hotel: Hotel ID, City, Hotel Name, Address, Telephone, Fax, Email, Rooms Available, Website, Price, Star Rating

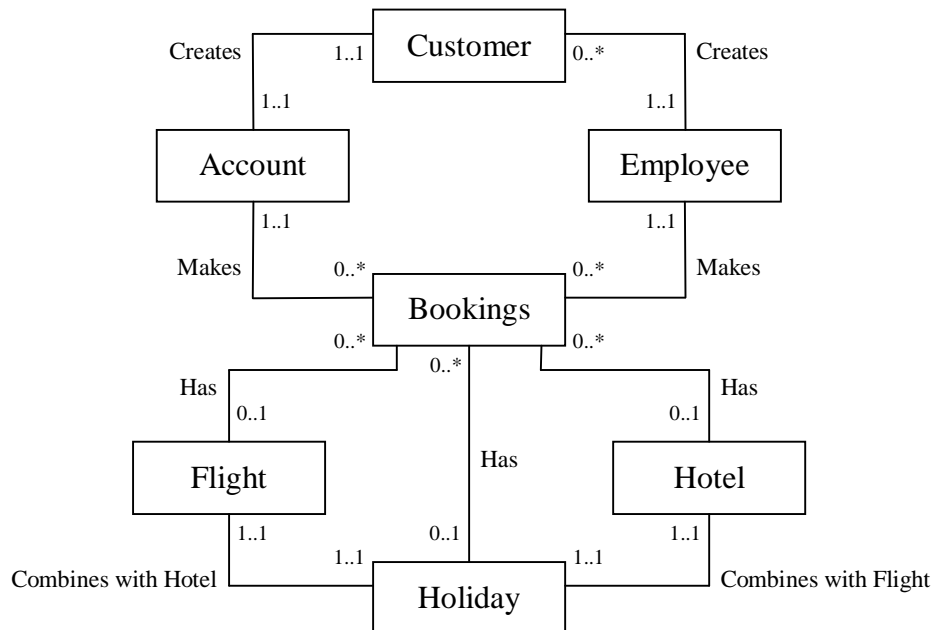
Holiday: Holiday ID, Hotel ID, Flight ID, Customer ID

Account: Customer ID, Booking ID

Booking: Booking ID, Departure Date, Return Date, Flight Code, Hotel ID, Customer ID, Seats, Rooms

Employee: Employee ID, Contact Phone

1.23 Entity Relationship Diagram



1.3 Tables and their Attributes

By looking at the entity attributes and the entity relationship diagram it is clear that there are a couple of entities that do not serve a definite purpose:

- Account
- Holiday

These on their own are not important as they can be easily searched using the existing tables - by having these tables it would just create redundant data.

1.31 Creating Tables

Below is a list of the tables which at this stage I would feel it necessary to create having discovered the entities minus the two entities above:

Customer: **Customer ID**, Surname, Forename, Address, Phone, Mobile, Email, DOB, Info

Flight: **Flight Code**, Departure City, Destination City, Flight Time, Airline, Airline Code, Airline Address, Airline Phone, Seats Available, Cost

Hotel: **Hotel ID**, City, Hotel Name, Address, Telephone, Fax, Email, Rooms Available, Website, Price, Star Rating

Employee: **Employee ID**, Contact Phone

Booking: **Booking ID**, Departure Date, Return Date, Date of Stay, **Flight Code**, **Hotel ID**, **Customer ID**, Seats, Room

**The Departure and Return dates, Flight code and seats would refer to a flight booking and Date of stay, Hotel ID and Room would refer to a hotel booking. Each night's stay in a hotel would be on a separate row.*

Key: **Red Text** = Primary Key, **Blue Text** = Foreign Key

1.32 Table Problems

Before I move onto normalisation of the tables I feel it is important to address certain problems that have occurred with the booking section of the database.

The problem splits into several points:

- Using one table will create many 'null' instances (as shown below) due to the possibilities of booking (Flight or Hotel)
- How to show a holiday booking or separate flight and hotel bookings
- Each hotel booking would create many rows

Below shows an example of a simple holiday booking for a 2 night stay using the single booking table:

Booking ID	Departure Date	Return Date	Date of Stay	Flight Code	Hotel ID	Customer ID	Seats	Rooms
123	13/01/07	15/01/07	Null	AI011	Null	762	3	Null
124	Null	Null	13/01/07	Null	R74	762	Null	2
125	Null	Null	14/01/07	Null	R74	762	Null	2

By looking at this it is clear that a table of this nature would not work as, even though all values are functionally dependent on the Booking ID making normalization easier, there are so many 'NULL' values and problems from this format as mentioned above that it would make more sense to use more than one table.

Solution 1:

- Use three tables:
- Flight_Booking
 - Hotel_Booking
 - Holiday Booking

When separate flights or hotels are booked they would be listed individually in their tables but when a flight and hotel are booked together (holiday) a 'Y' would be placed in the 'holiday' attribute meaning that the flight_booking_id and hotel_booking_id would be listed in the holiday_booking table so as it is easier to search for booked holidays.

Below shows an example using this method for a holiday being booked departing on the 13th January and returning on the 15th:

Flight Booking Table	F_booking_ID	Customer_ID	Date	Flight_Code	Seats	Hol
	123	762	13/01/07	AI011	3	Y
	124	762	15/01/07	AI012	3	Y

Hotel Booking Table	H_booking_ID	Customer_ID	Arrive_date	Depart_date	Hotel_code	Rooms	Hol
	125	762	13/01/2007	15/01/2007	R64	2	Y

Holiday Booking Table	F_booking_ID1	F_booking_ID2	H_booking_ID	Customer_ID
	123	124	125	762

Solution 2:

- Use three tables:
- Booking
 - Flight Booking
 - Hotel Booking

Every booking made would be listed in the booking table and have its own booking number. Each flight and hotel booked would then be listed in the flight or hotel booking table or both with the booking number being the same as the one in the booking table. A holiday attribute would also be added to the booking table to distinguish which bookings are holiday bookings.

Below shows 2 examples using this method for a holiday being booked departing on the 13th January and returning on the 15th and a hotel being booked separately arriving on the 17th January and leaving on the 18th January being booked by an employee:

Booking Table	Booking_ID	Customer_ID	Employee_ID	Hol
	172	762	NULL	Y
	173	NULL	133	N

Flight Booking Table	Booking_ID	Flight_Code	Seats	Depart_Date
	172	AI011	3	13/01/07
	172	AI012	3	15/01/07

Hotel Booking Table	Booking_ID	Hotel_ID	Arrival_Date	Depart_Date	Rooms
	172	R64	13/01/07	15/01/07	2
	173	S73	17/01/07	18/01/07	1

After considering both options I have decided that it would make it a lot easier to use Solution 2. This is because having one booking ID for each booking makes it a lot easier to trace each booking, rather than in solution 1 where there were flight booking numbers and hotel booking numbers, I will now be using this format for the tables in the normalization stage.

1.4 Normalization of Tables

After the extra tables I decided to use in section 1.32 I now have 7 tables:

- Customer
- Employee
- Flight
- Hotel
- Booking
- Flight Booking
- Hotel Booking

Some of these tables are already normalized which I will explain in section 1.41 and the rest I will talk through the normalization process in section 1.42.

1.41 Normalized Tables

The tables which are already in Boyce Codd 3rd Normal Form are Customer, Employee and Hotel. Below are examples of these.

Customer Table:

Customer_ID	Surname	Forename	Address	Phone	Mobile
101	Freeman	Ted	103 Talington Way, Romford, Essex, RM4 0UD	01273637283	07926372362
102	Dextor	Kelvin	74 Queens Street, Nottingham, NO14 8YD	01673635476	07835291827

Email	DOB	Info
ted@tedfreeman.com	07/04/67	Y
kelvin@kelvindextor.com	18/11/75	N

Employee Table:

Employee_ID	Contact_Phone
085	08458273728
054	08458279183

Hotel Table:

Hotel_ID	City	Hotel_Name	Address	Telephone	Fax
R01	Rome	Diclexiano	Via Gaeta 71, Rome, RM00185, Italy	06 48 90 0767	06 47 45891
B01	Barcelona	Grand Marina	Moll de Barcelona, Barcelona 08039, Spain	+34936039000	+34936039090

Email	Rooms_Available	Website	Price	Star_Rating
NULL	4	www.dioflexiano.com	100.00	5
infor@granmarinahotel.com	6	www.granmarinahotel.com	130.00	4

Key: Red Text = Primary Key

As you can see from the example entries in the 3 tables above these are all in Boyce Codd 3rd Normal Form as all non-key fields are dependent on the key field.

1.42 Tables to be Normalized

The Flight table needs to be normalized as there is repeating data with the Airline details so this is not in 1st Normal Form. To make this table 1st Normal Form I needed to separate the airline details into a separate table as shown below:

Flight Table:

Flight_Code	Depart_City	Arrival_City	Flight_Time	Airline_Code	Seats_Available	Cost
AI011	London	Rome	10:35	AI	5	50.00
SA662	Geneva	Manchester	06:35	SA	4	45.00

Airline Table:

Airline_Code	Airline	Address	Telephone
AI	Allitalia	Heathrow Airport, Hounslow, Middlesex, TW6 1HA	02088147700
SA	Swiss Air	World Business Centre, Newall Road, London Heathrow Airport, Hounslow, TW6 2RD	08456010956

Key: Red Text = Primary Key, Blue Text = Foreign Key

The booking tables (Booking Table, Flight Booking Table, Hotel Booking Table) need to be looked at to confirm if they are in 3rd normal form.

Booking Table:

Booking_ID	Customer_ID	Employee_ID	Hol
172	762	NULL	Y
173	NULL	133	N

Key: Red Text = Primary Key

Flight Booking Table:

Booking_ID	Flight_Code	Seats	Depart_Date
172	AI011	3	13/01/07
172	AI012	3	15/01/07

Hotel Booking Table:

Booking_ID	Hotel_ID	Arrival_Date	Depart_Date	Rooms
172	R64	13/01/07	15/01/07	2
173	S73	17/01/07	18/01/07	1

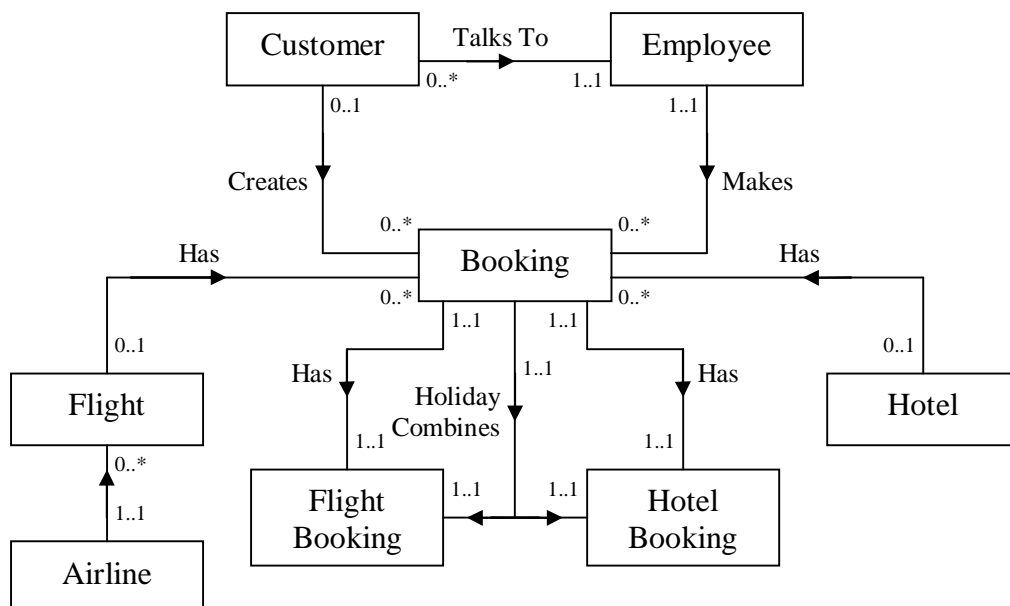
Key: Red Text = Joint Primary Key, Blue Text = Joint Primary Key / Foreign Key

By looking these tables it is clear that it is not as straight forward as the other tables. The Booking table has one primary key, which is the Booking_ID and all of the other attributes are functionally dependent on this, so this is in 3rd Normal Form.

However the Flight Booking Table and Hotel Booking Table do not have one single primary key as every attribute could have one or more of the same entries. Therefore it was necessary to create a complex key in the form of 2 primary keys. I decided to use the Booking_ID and the Flight_Code or Hotel_ID as this combination would never be the same. This is now in 3rd Normal Form as all other attributes are dependent on the primary keys.

1.5 The External Model

1.51 Entity Relationship Diagram



1.52 Use Case Relations

Below are the relations from the entity relationship diagram and the use cases from section 1.1 that they refer to.

Relation from and to	Refer to Use Cases
Customer > Employee	9, 10, 11
Customer > Booking	1, 2, 3, 4, 6, 7, 8
Employee > Booking	9, 10, 11
Airline > Flight	1
Flight > Booking	1
Hotel > Booking	2
Booking > Hotel Booking	7, 9
Booking > Flight Booking	6, 10
Booking > Flight and Hotel Booking	8, 11

1.6 The Internal Model

The following tables show the descriptions of the tables in this database:

Customer Table:

Field	Type	Null	Key	Default	Extra
Customer_ID	Int(4)	No	Primary	Null	Auto_increment
Surname	Varchar(20)	No		Null	
Forename	Varchar(20)	No		Null	
Address	Varchar	No		Null	
Phone	Varchar(13)	No		Null	
Mobile	Varchar(11)	Yes		Null	
Email	Varchar(30)	Yes		Null	
DOB	date	No		Null	
Info	char(1)	No		Y	

Employee Table:

Field	Type	Null	Key	Default	Extra
Employee_ID	Int(4)	No	Primary	Null	Auto_increment
Contact_Phone	Varchar(13)	No		Null	

Flight Table:

Field	Type	Null	Key	Default	Extra
Flight_Code	Varchar(5)	No	Primary	Null	
Depart_City	Varchar(20)	No		Null	
Arrival_City	Varchar(20)	No		Null	
Flight_Time	Time	No		Null	
Airline_Code	Char(2)	No		Null	
Seats_Available	Int(1)	No		Null	
Cost	Decimal	No		Null	

Airline Table:

Field	Type	Null	Key	Default	Extra
Airline_Code	Char(2)	No	Primary	Null	
Airline	Varchar(20)	No		Null	
Address	Varchar	No		Null	
Telephone	Varchar(13)	No		Null	

Hotel Table:

Field	Type	Null	Key	Default	Extra
Hotel_ID	Int(4)	No	Primary	Null	Auto_increment
City	Varchar(20)	No		Null	
Hotel_Name	Varchar(20)	No		Null	
Address	Varchar	No		Null	
Telephone	varchar(13)	No		Null	
Fax	varchar(13)	No		Null	
Email	varchar(30)	No		Null	
Rooms_Available	Int(1)	No		Null	
Website	varchar(30)	No		Null	
Price	decimal	No		Null	
Star_Rating	char(1)	No		Null	

Booking Table:

Field	Type	Null	Key	Default	Extra
Booking_ID	Int(4)	No	Primary	Null	Auto_increment
Customer_ID	Int(4)	Yes	Foreign	Null	
Employee_ID	Int(4)	Yes	Foreign	Null	
Hol	Char(1)	No		N	

Flight Booking Table:

Field	Type	Null	Key	Default	Extra
Booking_ID	Int(4)	No	Primary/Foreign	Null	
Flight_Code	Varchar(5)	No	Primary	Null	
Seats	Int(1)	No		Null	
Depart_Date	Date	No		Null	

Hotel Booking Table:

Field	Type	Null	Key	Default	Extra
Booking_ID	Int(4)	No	Primary/Foreign	Null	
Hotel_ID	Int(4)	No	Primary	Null	
Arrival_Date	Date	No		Null	
Depart_Date	Date	No		Null	
Rooms	Int(1)	No		Null	

3.1 SQL Commands

Shown here are the main SQL commands used throughout the database system.

3.11 Display search results for Flight/Hotel/Holiday

- (1)

```
$dquery = "select * from flight where depart_city='$dcity' and arrival_city='$sacity'";
$dresult = mysql_query($dquery);
if (mysql_num_rows($dresult) == 0)
{
echo "Sorry there are no flights available between $dcity and $sacity<br><br>";
echo 'Please <a href="index.php">click here</a> to return to the booking page';
}
else
{
```
- (2)

```
$rquery = "select * from flight where depart_city='$sacity' and arrival_city='$dcity'";
$rresult = mysql_query($rquery);
```
- (3)

```
$hquery = "select * from hotel where city='$city'";
$hresult = mysql_query($hquery);
```

This piece of code comes from the 'searchholiday' page but the same code is used in the single flight and hotel pages.

- (1) Searches the database for all flights where the depart city is equal to the depart city variable and the arrival city is equal to the arrival city variable forwarded from the main page and places the results in the '\$dresult' variable. On part 1 the if statement returns the message from the if statement if there are no results from the first part of the query.
- (2) Searches the database for the return part of the flight for all flights where the depart city is equal to the arrival city variable and the arrival city is equal to the depart city variable forwarded from the main page and places the results in the '\$rresult' variable.
- (3) Selects all details from the hotel table where the city is equal to the city variable forwarded from the main page and places the result in the '\$hresult' variable.

3.12 Get Star Image and Hotel Image

- (1) `$hotelid=$_GET['id'];`
- (2) `$q="SELECT star FROM hotel WHERE hotel_id = '$hotelid';`
`$r=mysql_query($q);`
`$row=mysql_fetch_row($r);`
- (3) `header("Content-Type: image/jpeg");`
`echo $row[0];`

- (1) Place the ID from the 'Get' parameter from the select holiday / hotel page into the '\$hotelid' variable
- (2) Selects star or image from the hotel table where hotel ID is equal to the '\$hotelid' variable – this should and must only return one result and place this in '\$row' variable
- (3) State the header type of the image and then echo the result as an image from row 0 (the first row)

3.13 Process Customer / Employee Login

- (1) `$query = "select email, password from customer where email='$email' and password='$password';`
`$result = mysql_query($query);`
- (2) `$query = "select employee_id from employee where employee_id='$emp';`
`$result = mysql_query($query);`

- (1) Select the email and password from the customer table where the values are the same as those stored in the variables from the login page (This will see if they have registered)
- (2) For an employee login request the employee_id will be selected from the employee table that is the same as the stored variable from the employee login to see if it exists.

3.14 Select Flight / Hotel / Holiday Details

- (1) `$dquery = "select * from flight where flight_code='$dflight';`
`$dresult = mysql_query($dquery);`
- (2) `$dquery2 = "select cost from flight where flight_code='$dflight';`
`$dresult2 = mysql_query($dquery2);`
`$dcost = mysql_result($dresult2,$i,"cost");`

- (3) `$dexisting = "select sum(bseats) as total from flight_booking where flight_code='$dflight' and depart_date='$ddate'";`
`$dres_existing = mysql_query($dexisting);`
`$db_seats = mysql_result($dres_existing,$i,"total");`
`if ($db_seats == "")`
`{`
`$db_seats = "0";`
`}`
`else {`
`}`
- (4) `$dseats_a = "select(select seats from flight where flight_code='$dflight')-($db_seats) as total";`
`$dresult3 = mysql_query($dseats_a);`
`$da_seats=mysql_result($dresult3,$i,"total");`
- (5) `$rquery = "select * from flight where flight_code='$rflight'";`
`$rresult = mysql_query($rquery);`
- (6) `$rquery2 = "select cost from flight where flight_code='$rflight'";`
`$rresult2 = mysql_query($rquery2);`
`$rcost = mysql_result($rresult2,$i,"cost");`
- (7) `$rexisting = "select sum(bseats) as total from flight_booking where flight_code='$rflight' and depart_date='$rdate'";`
`$rres_existing = mysql_query($rexisting);`
`$rb_seats = mysql_result($rres_existing,$i,"total");`
`if ($rb_seats == "")`
`{`
`$rb_seats = "0";`
`}`
`else {`
`}`
- (8) `$rseats_a = "select(select seats from flight where flight_code='$rflight')-($rb_seats) as total";`
`$rresult3 = mysql_query($rseats_a);`
`$ra_seats=mysql_result($rresult3,$i,"total");`
- (9) `$hquery = "select * from hotel where hotel_id='$shotel'";`
`$hresult = mysql_query($hquery);`
- (10) `$hquery2 = "select price from hotel where hotel_id='$shotel'";`
`$hresult2 = mysql_query($hquery2);`
`$hcost = mysql_result($hresult2,$i,"price");`
- (11) `$hexisting = "select sum(brooms) as total from hotel_booking where hotel_id='$shotel' and ((arrival_date <= '$ddate' and depart_date > '$ddate') OR (arrival_date < '$rdate' and depart_date >= '$rdate') OR (arrival_date >= '$ddate' and depart_date <= '$rdate'))";`
`$hres_existing = mysql_query($hexisting);`
`$b_rooms = mysql_result($hres_existing,$i,"total");`
`if ($b_rooms == "")`
`{`
`$b_rooms = "0";`
`}`
`else {`
`}`
- (12) `$rooms_a = "select(select rooms from hotel where hotel_id='$shotel')-($b_rooms) as total";`
`$hresult3 = mysql_query($rooms_a);`
`$a_rooms=mysql_result($hresult3,$i,"total");`

```
(13) $dateq = "select datediff('$rdate','$ddate') as total";
    $dateresult = mysql_query($dateq);
    $nodays = mysql_result($dateresult,$i,"total");
```

This piece of code comes from the 'selectholiday' page but the same code is used in the single flight and hotel pages.

- (1) Select all from the flight table where the flight_code is equal to the '\$dflight' (departure flight) variable and store the results in the \$dresult variable
- (2) Select the cost from flight where the flight_code is equal to the \$dflight variable and place the single result in the \$dcost variable
- (3) Select the total sum from the flight booking table where the flight_code is equal to the \$dflight variable and the depart_date is equal to the \$ddate variable. Place this single result in the \$db_seats variable. The if statement detects if no results were found and if so puts '0' as the \$db_seats variable
- (4) This selects as total the total amount of seats that would be available on this flight minus the amount of booked seats found out from part (3) and places the result in the \$da_seats variable
- (5) Reversed action for the return flight as seen in part (1)
- (6) Reversed action for the return flight as seen in part (2)
- (7) Reversed action for the return flight as seen in part (3)
- (8) Reversed action for the return flight as seen in part (4)
- (9) Select all from the hotel table where the hotel_id is equal to the '\$hotel' variable and store the results in the \$hresult variable
- (10) Select the price from the hotel table where the hotel_id is equal to the \$hotel variable and place the single result in the \$hcost variable
- (11) Select the total sum of rooms from the hotel booking table where the hotel_id is equal to the \$hotel variable and:
 - (11.1) A booking exists where the arrival date is before \$ddate variable and depart date is after \$rdate variable
 - (11.2) A booking exists where the arrival date is before the \$rdate variable and depart date is after the \$rdate variable
 - (11.3) A booking exists where the arrival date is after the \$ddate variable and the depart date is before the \$rdate variable

* This checks for all of the possible ways that bookings could overlap the booking currently trying to be made.
The if statement detects if no results were found and if so puts '0' as the \$b_rooms Variable
- (12) This selects as total the total amount of rooms that would be available at this hotel minus the amount of booked rooms found out from part (11) and places the result in the \$a_rooms variable
- (13) This statement works out the number of days that the booking lasts for so that the total cost can be worked out. It selects the day difference between the \$rdate variable and the \$ddate variable and places the result in the \$nodays variable

3.15 Book Flight / Hotel / Holiday

```
(1) $query = "select customer_id from customer where email='$email'";
    $result = @mysql_query ($query);
```

```

$C_id=mysql_result($result,$i,"customer_id");

(2) $query = "insert into booking (customer_id, date_booked, hol) values ('$C_id', '$timestamp', 'Y')";
$result = @mysql_query ($query);

if ($result) {
(3) $query2 = "select booking_id from booking where customer_id='$C_id' and
date_booked='$timestamp'";
$result2 = @mysql_query ($query2);
$b_id=mysql_result($result2,$i,"booking_id");

(4) $query3 = "insert into flight_booking (booking_id, flight_code, bseats, depart_date)
values ('$b_id', '$dflight', '$seats', '$ddate')";
$result3 = @mysql_query ($query3);

(5) $query5 = "insert into hotel_booking (booking_id, hotel_id, arrival_date, depart_date, brooms)
values ('$b_id', '$shotel', '$ddate', '$rdate', '$rooms')";
$result5 = @mysql_query ($query5);

(6) $query4 = "insert into flight_booking (booking_id, flight_code, bseats, depart_date)
values ('$b_id', '$rflight', '$seats', '$rdate')";
$result4 = @mysql_query ($query4);

(7) $query6 = "select name from hotel where hotel_id='$shotel'";
$result6 = @mysql_query ($query6);
$shotelname=mysql_result($result6,$i,"name");

```

This piece of code comes from the ‘bookholiday’ page but the same code is used in the single flight and hotel pages.

- (1) Select the customer ID from the customer table where the email is equal to the \$email variable and place it in the \$C_id variable
- (2) Insert into the booking table the \$C_id and \$timestamp variables and ‘Y’ or ‘N’ will be stored in the ‘hol’ field depending on if a holiday has been booked (\$timestamp comes from the timestamp calculated in PHP, shown in section 3.2). If booked by an employee the \$emp variable will be stored as employee_no and the customer_id will be null.
- (3) Select the booking ID that was created from inserting the data in part (2) by selecting the customer ID from the \$C_id variable (or the employee_id from the \$emp variable) and the date_booked field is equal to the \$timestamp variable and place the result in the \$b_id variable
- (4) Insert into the flight booking table the \$B_id, \$dflight, \$seats and \$ddate variables
- (5) Insert into the hotel booking table the \$B_id, \$shotel, \$ddate, \$rdate and \$rooms variables
- (6) Insert into the flight booking table the \$B_id, \$rflight, \$seats and \$rdate variables
- (7) Select the name from the hotel table where the hotel_id is equal to the \$shotel variable and place in the \$shotelname variable; to be used to display the booking confirmation details

3.2 System Description

3.21 Page Layout:


```

(2) while ($drow = @ mysql_fetch_array($dresult))
{
print      "\n<tr>\n\t<td>{ $drow["flight_time"]}</td>" .
      ...
      "\n\t<td>{ $seats }</td>" .
      "\n\t<td><input type=\"radio\" name=\"dflight\" value=\"\".$drow{ 'flight_code' }.\"></td>";
      "\n</tr>";
}
print "\n</table>\n <br><br>";

(3) while ($hrow = @ mysql_fetch_array($hresult))
{
print      "\n\t<span class=\"textsubhead\">{ $hrow["name"]}</span>" ;
echo '      <div id="left">';
print      '' .
      "\n\t<br><br>Address:<br>" .
      ...
echo      '</div><div id="right">';
print      '';
echo      '</div><div id="middle">';
print      "\n\t<br><br>{ $hrow["address"]}" .
      ...
print      "<br><br>\n\t</div>";
}

```

(1) This creates the structure of the table in which to display the flight results tables. The lines

represent the table rows

(2) The while loop will place each row of the returned database search in the \$drow variable until it has shown all of the results. The 'print' lines refer to the rows in part (1), the flight time is taken from the \$drow variable whereas the seats is taken directly from the variable

from the previous page. The next line creates a radio button to select this particular row form the database result

(3) This code is used for displaying the hotel results. It uses the same while loop as part (2), however this formatting ensures each result is shown in its own section rather than in a table.

The image lines connect to the 'getstar' and 'getpic' sections to get the correct image (see section 3.12)

3.32 Detect Login

```

(1) if
($_SESSION['loggedin']==1)
{if
( $from=='http://localhost/final/searchflight.php')
{include ('selectflight_cust.php');
}
else
{
include ('login.php');
}
}

```

(1) This is an if statement to detect if the session 'loggedin' is set to 1 then to carry on as usual,

but if not then include the login page

3.33 Process Login

```
(1) if (mysql_num_rows($result) == 0)
    {
        include ('login_error.php');
    }
    else
    {
        $_SESSION['loggedin']=1;
        if ( $from=='http://localhost/final/searchflight.php' )
        {
            $page='selectflight_cust.php';
        }
        include ($page);
    }
}
```

(1) This if statement detects if there were no results returned from the database search (see section 3.13). It includes the login_error page if there are no results otherwise it includes the page that you would be viewing next and sets the session 'loggedin' to '1'

3.34 Select Flight / Hotel / Holiday

The main results shown here uses the same code to display the results as section 3.31.

```
(1) if ($seats <= $da_seats)
    { ... }
    else
    {
        echo 'Sorry...';
    }

(2) $dtcost = $dcost * $seats;

(3) $tholcost = $dtcost+$rtcost+$htrcost;
    $discount = $tholcost*0.95;
```

(1) This code is placed around each section (outward flight, return flight and hotel) which checks

if the seats or rooms asked for is less than or equal to the seats or rooms available (as found in the search in section 3.14), if it is then it continues to display the results, if not then a message will be displayed

(2) This works out the cost of each section by multiplying the cost by the number of seats or rooms

(3) Holiday Only. This works out the total cost of the holiday (adding the departure flight, return

flight and hotel cost) and then multiplies this by 0.95 to get the 5% which is then subtracted

from the total holiday cost

3.35 Book Flight / Hotel / Holiday

This section uses the same syntax as shown in section 3.31 for displaying the conformation of the booking.

3.4 Indexing

An index in a database is very similar to an index in a book. It allows a database to be searched quickly and efficiently without having to browse through the entire file each time you need to find a record.

As with book indexes, database indexes contain items ordered and each item contains the location or locations where the item can be found. Two types of file exist in a database index; a data file containing the full records and an index file containing the index records. The index structure for this file has a particular search key and contains the key value of the record and the address of this record, this search key can consist of one or more fields.

In the case of this database system it could have many more records stored before long that needs to be accessed. There are only a few items in both the hotel tables and the flight and airline tables at the current time but in the future there would be many entries from many more cities and airlines making searches from each of the tables a lot longer to complete.

Indexing would be very useful in this situation as each table could be indexed by city in the hotel table for example. This would mean that each search would look at the city in the index which would show what part of the hotel table has the records from that particular city making the search a lot easier and the results would be returned a lot quicker. The same idea could be applied to the flight table. The flight table could be indexed by destination airport where the index would contain the address of the records relating to all of the flights that go to that destination. This would make searching for flights to a particular destination a lot easier and quicker.

So indexing could be applied to this database system if it was to increase in size very successfully.

3.5 Denormalisation

Denormalisation is a technique that is used to improve the performance of a database so as to speed up queries for example.

It works by undoing some of ways that normalisation changed the database when it was in the design phase. One such techniques used at normalisation is splitting data up into different tables so that joins can be used to minimise redundancy can end up causing queries to take longer to find and so denormalisation reverses this to try and improve performance.

It should be noted, however, that denormalisation introduces 'controlled' redundancy so the database would not be like it was before it was normalised.

Performance of this database could be compromised when there are many bookings in the bookings table and searching the 3 booking tables could take up valuable time, therefore it might be needed to denormalise these tables into 2 or even 1 table if necessary. By doing this it would minimise the need for joins between the 3 booking tables.