L14: Database-Dependent Websites

CS1106/CS6503: Intro to Relational Databases

Dr Kieran T. Herley

Semester One, 2023-24

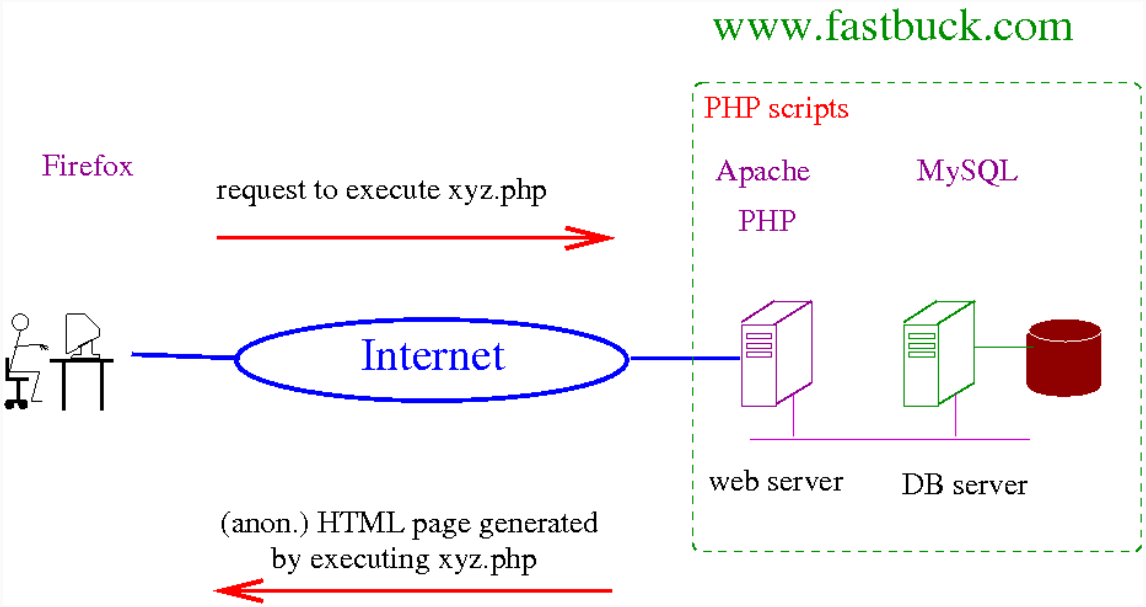
School of Computer Science & Information Technology
University College Cork

Summary

Integration of databases and websites. Python scripts for managing HTML-form and DB interaction. Jolly Roger case study. Summary of JRA's DB. Sample queries.

Typical E-Commerce Architecture

- Most e-commerce sites are are built around a DB; typically DB houses info. on stock, customers, billing etc.



- User's interaction with site mediated by scripts (e.g. in Python/PHP) executing on the server
- Scripts can "talk to" DB to extract/insert/modify information

Python and SQL

- Can "embed" SQL queries within Python scripts and arrange to execute script (and query) in response to (say) submit event for HTML form.
- Script takes care of:
  - establishing DB connection etc.
  - dispatching query to DB server
- (Script may also "process" query result and e.g. build a "response page" (HTML) out of result data.)

Suggestion box example

Implement simple on-line suggest box for CS1106.

Elements:

- (Server) Simple DB table:

  `suggestions(id, comment, users, email, date)`

- (Server) SQL query to insert a fresh comment:

```
INSERT INTO suggestions VALUES
    ("12345", "Wow!", "KTH", "kth@ucc.ie", "2017-11-03")
```

- (Server) Python script including query triggered when form submitted (over)
- (Client) HTML form with text box (for text of suggestion) and submit button (also text fields for user details).

Template for Python script

record_suggestion.py

Input: *Details of the customer's contact details and suggestion from HTML form*
Output: *HTML page thanking customer for suggestion*

Operation:
Submit following SQL query to DB server:

```
INSERT INTO suggestions VALUES ( . . .);
```

Generate response page summarizing customer's suggestion and acknowledging same.

How this works

Client

1. User views form in `suggestion_box.html` and completes it.

Server

How this works

Client

1. User views form in `suggestion_box.html` and completes it.
2. User clicks submit.

Server

How this works

Client

1. User views form in `suggestion_box.html` and completes it.
2. User clicks submit.
3.

Server

Script `record_suggestion.py`:

3.1 Extracts form data

3.2 Embeds data in SQL command– to insert suggestion details in DB table `suggestions`

3.3 Submits SQL command to DB server

3.4 Generates response page, thanking user for suggestion

How this works

Client

1. User views form in `suggestion_box.html` and completes it.
2. User clicks submit.
3.
4. User's browser displays response

Server

Script `record_suggestion.py`:

3.1 Extracts form data

3.2 Embeds data in SQL command– to insert suggestion details in DB table `suggestions`

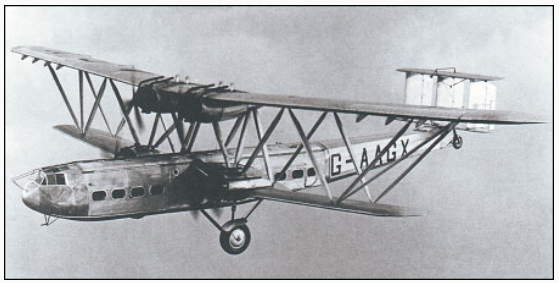
3.3 Submits SQL command to DB server

3.4 Generates response page, thanking user for suggestion

Jolly Roger Airlines

Example of website-database interaction

- Low-cost airline *Jolly Roger Airlines* (JRA) in need of on-line reservation system



  - DB to house data (services, availability, prices, bookings etc.)
  - Scripts to handle various stages of booking process

Typical booking session with JRA website

Client

1. Customer submits trip details using form

Server

Typical booking session with JRA website

Client

1. Customer submits trip details using form
2.

Server

Script `check_services` interrogates DB for suitable services, "packages" results as HTML response page and returns this to client

Typical booking session with JRA website

Client

1. Customer submits trip details using form
2.
3. Customer views response and chooses desired flights

Server

Script `check_services` interrogates DB for suitable services, "packages" results as HTML response page and returns this to client

Typical booking session with JRA website

Client	Server
1. Customer submits trip details using form	
2.	Script `check_services` interrogates DB for suitable services, "packages" results as HTML response page and returns this to client
3. Customer views response and chooses desired flights	
4.	Script `check_availability` interrogates DB for price and availability of selected flights and packages results as HTML page and returns this to client.

Typical booking session with JRA website

Client	Server
1. Customer submits trip details using form	
2.	Script `check_services` interrogates DB for suitable services, "packages" results as HTML response page and returns this to client
3. Customer views response and chooses desired flights	
4.	Script `check_availability` interrogates DB for price and availability of selected flights and packages results as HTML page and returns this to client.
5. Customer submits payment details.	

Typical booking session with JRA website

Client	Server
1. Customer submits trip details using form	
2.	Script `check_services` interrogates DB for suitable services, "packages" results as HTML response page and returns this to client
3. Customer views response and chooses desired flights	
4.	Script `check_availability` interrogates DB for price and availability of selected flights and packages results as HTML page and returns this to client.
5. Customer submits payment details.	
6.	Script `process_booking` records details of booking, updates flight

Typical booking session with JRA website

Client	Server
1. Customer submits trip details using form	
2.	Script `check_services` interrogates DB for suitable services, "packages" results as HTML response page and returns this to client
3. Customer views response and chooses desired flights	
4.	Script `check_availability` interrogates DB for price and availability of selected flights and packages results as HTML page and returns this to client.
5. Customer submits payment details.	
6.	Script `process_booking` records details of booking, updates flight

Some Simplifying Assumptions

- One-way, point to point, single-seat bookings only
- Every service operates daily, 365 days a year

A Database Design

- A DB schema:

```
services(code, origin, destination, departure,
         duration, schedule)
flights(code, date, capacity, availability, price)
airports(code, name, timezone)
customers(customer_id, first_name, last_name,
        email, password)
bookings(booking_code, customer, service,
        date, credit_card)
```

Design cont'd

- ```
  services(code, origin, destination, departure,
           duration, schedule)
  flights(code, date, capacity, availability, price)
  ```
  ```

Design cont'd

- ```
  services(code, origin, destination, departure,
           duration, schedule)
  flights(code, date, capacity, availability, price)
  ```
- Each *service* e.g. `JR822` from `ORK` to `CDG`, operates daily at the same time
  - origin and destination represented by airport codes, e.g. `ORK`
  - all times expressed GMT (time-zone adjustment possible using `airports` time-zone values)
  ```

Design cont'd

- ```
  services(code, origin, destination, departure,
           duration, schedule)
  flights(code, date, capacity, availability, price)
  ```
- Each *service* e.g. `JR822` from `ORK` to `CDG`, operates daily at the same time
  - origin and destination represented by airport codes, e.g. `ORK`
  - all times expressed GMT (time-zone adjustment possible using `airports` time-zone values)
- Distinct *flight* for each service-date combination, e.g. JR822 on 1 Dec 2012
  - `capacity` reflects num. of seats on plane (fixed)
  - `availability` reflects num. of seats left
  - fixed `price` for each flight
  ```

Design cont'd

- `customers(customer_id, first_name, last_name, email, passwor`
- Repeat customers can re-use same customer id

Design cont'd

- `flights(code, date, capacity, availability, price)`
  `customers(customer_id, first_name, last_name, email, pa`
  `bookings(booking_code, customer, service, date, credit_`
- `bookings` models relationship of which customers are booked on which flights
  - `booking_code` unique alphanumeric code for this booking
  - `customer` customer's id number
  - `flight` flight's code

A query checking for suitable services

- Imagine customer is interested in flights from Cork(ORK) to Paris(CDG) (extracted from form data submitted by customer)

A query checking for suitable services

- Imagine customer is interested in flights from Cork(ORK) to Paris(CDG) (extracted from form data submitted by customer)
- Query

```
SELECT *
FROM services
WHERE
    services . origin  =  ORK  AND
    services . destination  =  CDG ;
```

- Script "embeds" customer requirements (boxed elements) into query
- Selection of date-specific flights comes at next stage

A Query cont'd

check_services.py

Input: *Details of customer's requirements as entered in HTML form*

Output: *HTML page summarizing suitable options*

A Query cont'd

check_services.py

Input: *Details of customer's requirements as entered in HTML form*

Output: *HTML page summarizing suitable options*

Operation:

1. Extract form data detailing customer requirements
2. Construct SQL query (incorporating reqs.) and submit to DB server

```
SELECT * FROM services ...
```

3. Take query result received from DB and generate HTML response page therefrom

Checking Availability And Price

- Similarly script `check_availability` would be built around the following query

```
SELECT price, . . .
FROM
    services JOIN flights
    ON services.code = flight .code
WHERE
    services . origin = ORK
    AND services. destination = CDG
    AND flight.date = 1/12/2012
    AND availability > 0;
```

-
  - Particularizes search to specific date
  - Displays prices for available flights only

Recording Booking

- Script `process_booking` would be built around a number of statements
  - Record the customer's details

```
INSERT INTO customers VALUES
    ( customer's id , . . . );
```

Recording Booking

- Script `process_booking` would be built around a number of statements
  - Record the customer's details

```
INSERT INTO customers VALUES
    ( customer's id , . . . );
```

  - Record the details of this booking

```
INSERT INTO bookings VALUES
    ( booking code , . . . );
```

Recording Booking

- Script `process_booking` would be built around a number of statements
  - Record the customer's details

```
INSERT INTO customers VALUES
    ( customer's id , . . . );
```

  - Record the details of this booking

```
INSERT INTO bookings VALUES
    ( booking code , . . . );
```

  - Update the availability information for this flight

```
UPDATE flights
SET availability = availability − 1
WHERE flight.code = JR666 AND
    flight .date = 1/12/2012 ;
```

A Potential Snag . . .

Customer 1 **Customer 2**

A Potential Snag . . .

	Customer 1	Customer 2
09:00	Customer 1 queries ORK-CDG flights for 1 December 2012	

A Potential Snag . . .

	Customer 1	Customer 2
09:00	Customer 1 queries ORK-CDG flights for 1 December 2012	
09:01		Customer 2 queries ORK-CDG flights for 1 December 2012

A Potential Snag . . .

	Customer 1	Customer 2
09:00	Customer 1 queries ORK-CDG flights for 1 December 2012	
09:01		Customer 2 queries ORK-CDG flights for 1 December 2012
09:02	Customer 1 views response indicating one seat left; quickly tries to book flight	

A Potential Snag . . .

	Customer 1	Customer 2
09:00	Customer 1 queries ORK-CDG flights for 1 December 2012	
09:01		Customer 2 queries ORK-CDG flights for 1 December 2012
09:02	Customer 1 views response indicating one seat left; quickly tries to book flight	
09:03		Customer 2 views response indicating one seat left; quickly tries to book flight

A Potential Snag . . .

	Customer 1	Customer 2
09:00	Customer 1 queries ORK-CDG flights for 1 December 2012	
09:01		Customer 2 queries ORK-CDG flights for 1 December 2012
09:02	Customer 1 views response indicating one seat left; quickly tries to book flight	
09:03		Customer 2 views response indicating one seat left; quickly tries to book flight
09:04	Customer 1's booking is reflected in DB	

A Potential Snag . . .

	Customer 1	Customer 2
09:00	Customer 1 queries ORK-CDG flights for 1 December 2012	
09:01		Customer 2 queries ORK-CDG flights for 1 December 2012
09:02	Customer 1 views response indicating one seat left; quickly tries to book flight	
09:03		Customer 2 views response indicating one seat left; quickly tries to book flight
09:04	Customer 1's booking is reflected in DB	
09:05		??????

Need some mechanism to prevent "simultaneous" bookings interfering with one another and potentially corrupting the DB

Notes and Acknowledgements

The photograph of the ancient aircraft is from the website `www.aviastar.org`. The Jolly Roger is from Wikicommons.