



Informal School Of IT

Competences needed in Software Testing

Handout Material





Contents

| | | |
|-------|---------------------------------------|----|
| 1 | Introduction | 6 |
| 1.1 | What is testing? | 6 |
| 1.2 | Testing process..... | 6 |
| 1.2.1 | Requirements Analysis..... | 7 |
| 1.2.2 | Test Planning..... | 7 |
| 1.2.3 | Test Analysis and Design | 8 |
| 1.2.4 | Test Execution | 8 |
| 1.2.5 | Test Reporting..... | 9 |
| 1.2.6 | Test Closure..... | 9 |
| 1.3 | QA vs. QC vs. Testing..... | 10 |
| 2 | Bug Management..... | 11 |
| 2.1 | What is a software bug? | 11 |
| 2.2 | Bug Reporting | 11 |
| 2.3 | Bug Reporting Rules..... | 12 |
| 3 | Requirements Analysis..... | 15 |
| 3.1 | Requirements Fundamentals..... | 15 |
| 3.1.1 | Functional Requirements..... | 15 |
| 3.1.2 | Non-Functional Requirements..... | 16 |
| 3.2 | Requirements Clarification | 16 |
| 3.2.1 | What questions should be asked? | 17 |
| 3.3 | Requirements breakdown | 18 |
| 4 | Test Analysis and Design..... | 19 |
| 4.1 | Test Analysis..... | 19 |
| 4.2 | Test Design..... | 20 |
| 4.2.1 | Test Design Techniques..... | 20 |
| 4.2.2 | Equivalence Class Partitioning | 21 |
| 4.2.3 | Boundary Value Analysis..... | 22 |
| 4.2.4 | Test Case | 23 |
| 4.2.5 | Test management tool - TestLink..... | 24 |
| 4.3 | Test Data | 24 |



| | | |
|------|---|----|
| 4.4 | Test Environment..... | 25 |
| 5 | Windows/Dos command line..... | 28 |
| 5.1 | File system structure..... | 28 |
| 5.2 | Help commands | 29 |
| 5.3 | Directories commands | 30 |
| 5.4 | Files commands..... | 31 |
| 5.5 | Other commands | 32 |
| 5.6 | Linux Commands..... | 33 |
| 6 | Testing Types | 35 |
| 6.1 | Black-Box vs. White-box testing types..... | 35 |
| 6.2 | Static vs. Dynamic testing types..... | 35 |
| 6.3 | Functional vs. Non-functional testing types | 36 |
| 6.4 | Manual vs. Automated testing types..... | 36 |
| 6.5 | Exploratory Testing | 37 |
| 7 | Testing Levels | 38 |
| 7.1 | Unit testing..... | 38 |
| 7.2 | Component testing | 38 |
| 7.3 | Integration testing | 39 |
| 7.4 | System testing..... | 39 |
| 7.5 | System integration testing (SIT)..... | 40 |
| 7.6 | (User) Acceptance testing..... | 40 |
| 8 | Test Execution..... | 42 |
| 8.1 | Bug (fix) validation | 42 |
| 8.2 | Regression Testing | 42 |
| 8.3 | Test execution good practices | 43 |
| 9 | Test Reporting..... | 45 |
| 10 | Testing specifics - Backend testing..... | 46 |
| 10.1 | Backend basics | 46 |
| 10.2 | Web Services..... | 48 |
| 10.3 | Backend Testing | 48 |
| 10.4 | SOAP vs REST..... | 50 |



| | | |
|---------|--|----|
| 10.5 | SOAP UI Tool | 51 |
| 11 | MySQL | 53 |
| 11.1 | Database | 53 |
| 11.2 | Structured Query Language (SQL)..... | 54 |
| 11.3 | Database main operations CRUD..... | 54 |
| 11.4 | Data Definition Language (DDL)..... | 54 |
| 11.4.1 | Data Types..... | 54 |
| 11.4.2 | Primary Key | 55 |
| 11.4.3 | Data Definition Language..... | 55 |
| 11.5 | Data Manipulation Language (DML) | 57 |
| 11.5.1 | SELECT | 57 |
| 11.5.2 | Arithmetic Operations | 58 |
| 11.5.3 | NULL Value..... | 58 |
| 11.5.4 | Column Aliases | 58 |
| 11.5.5 | Concatenation..... | 59 |
| 11.5.6 | Literals..... | 59 |
| 11.5.7 | Removing Duplicates..... | 59 |
| 11.5.8 | UNION, INTERSECT, EXCEPT | 60 |
| 11.5.9 | Limit Selected Rows | 60 |
| 11.5.10 | Other Comparison..... | 61 |
| 11.5.11 | Comparing with NULL | 61 |
| 11.5.12 | Logical Operators | 62 |
| 11.5.13 | ORDER BY | 62 |
| 11.5.14 | INSERT | 63 |
| 11.5.15 | UPDATE | 63 |
| 11.5.16 | DELETE..... | 64 |
| 11.6 | Multiple Tables..... | 64 |
| 11.6.1 | Cartesian Product..... | 65 |
| 11.6.2 | Equijoins..... | 65 |
| 11.6.3 | INNER JOIN | 66 |
| 11.6.4 | Nested SELECT..... | 66 |



| | | |
|---------|--|----|
| 11.6.5 | Nested SELECT with Aliases | 67 |
| 11.6.6 | EXISTS | 67 |
| 11.6.7 | Group Functions..... | 67 |
| 11.6.8 | Group Functions..... | 68 |
| 11.6.9 | AVG and SUM..... | 68 |
| 11.6.10 | MIN and MAX..... | 69 |
| 11.6.11 | COUNT..... | 69 |
| 11.6.12 | Group Function and NULL..... | 69 |
| 11.6.13 | Creating Groups of Data..... | 70 |
| 11.6.14 | GROUP BY..... | 70 |
| 11.6.15 | Grouping by More Columns..... | 71 |
| 11.6.16 | HAVING | 71 |
| 11.6.17 | Standard Functions | 72 |
| 11.7 | Transactional Control Language (TCL) | 73 |
| 11.7.1 | ACID Principles | 73 |
| 11.7.2 | Transactions | 73 |
| 11.7.3 | Stored Procedures..... | 74 |
| 12 | JavaScript (JS)..... | 74 |
| 12.1 | DHTML..... | 74 |
| 12.2 | Introduction to JavaScript..... | 75 |
| 12.2.1 | What is JavaScript? | 75 |
| 12.2.2 | JavaScript Advantages..... | 76 |
| 12.2.3 | Using JavaScript Code | 77 |
| 12.2.4 | JavaScript -When is Executed?..... | 77 |
| 12.2.5 | Using External Script Files | 78 |
| 12.3 | JavaScript Syntax..... | 79 |
| 12.3.1 | Data Types..... | 79 |
| 12.3.2 | Operations..... | 80 |
| 12.3.3 | Examples | 81 |
| 12.3.4 | Statements..... | 82 |
| 12.3.5 | Functions..... | 84 |



| | | |
|--------|----------------------------------|----|
| 12.4 | Document Object Model (DOM)..... | 85 |
| 12.4.1 | Accessing Elements..... | 86 |
| 12.4.2 | DOM Manipulation | 86 |
| 12.4.3 | The HTML DOM Event Model | 88 |
| 12.5 | Built-in Browser Objects | 89 |
| 12.5.1 | DOM Hierarchy - Example..... | 90 |
| 12.5.2 | Document and Location | 91 |
| 12.5.3 | The Math Object | 92 |
| 12.5.4 | The Date Object | 92 |
| 12.5.5 | Timers: setTimeout()..... | 93 |
| 12.5.6 | String Functions | 93 |
| 12.5.7 | Timer - Example | 94 |
| 12.6 | Debugging JavaScript | 94 |
| 12.6.1 | JavaScript Console Object | 95 |
| 13 | References | 96 |



1 Introduction

1.1 What is testing?

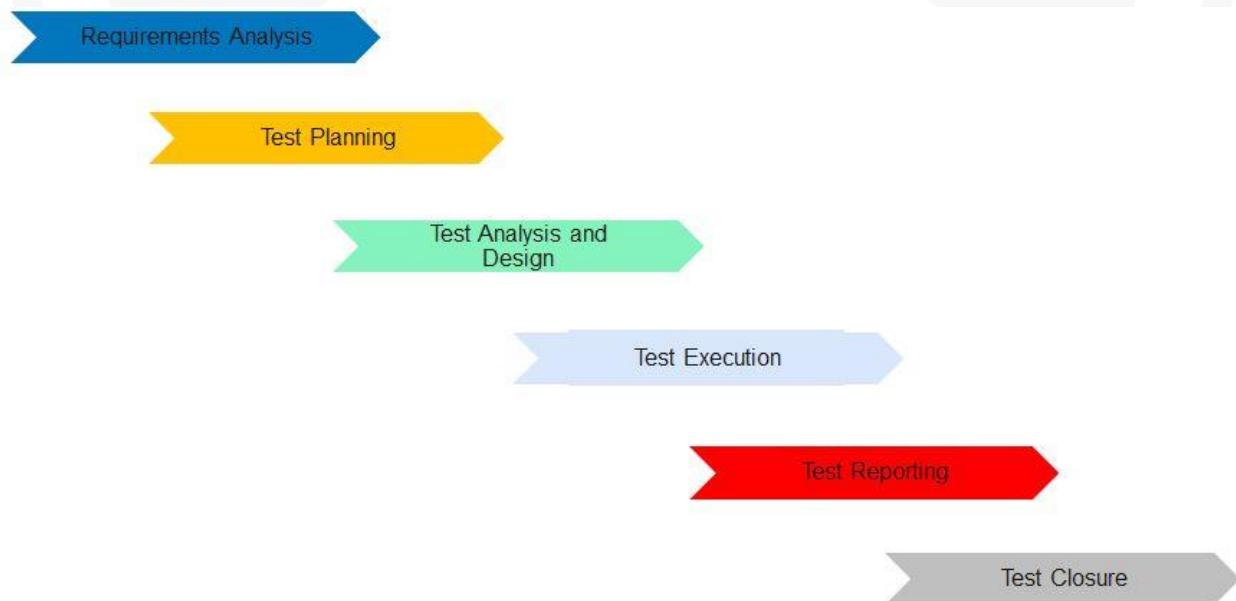
Software testing is the process of executing a program or application with the intent of finding any system malfunction.

The testing takes place throughout the software development life cycle.

The testing should answer some of the following questions:

- Does the software meet the requirements?
- Do all features work correctly?
- Are there defects in the software?
- Do we have enough information for decision-making?

1.2 Testing process





1.2.1 Requirements Analysis

Why?

- To understand the product requirements, what the client wants
- To raise any requirements inconsistencies as soon as possible
- To refine the scope of testing

How?

- The requirements are read and analyzed
- Clarify all requirements documents

1.2.2 Test Planning

Why?

- Identify all activities, dependencies, resources and timeline
- To make sure that the testing is completed in agreed time and quality

How?

- Define the scope of testing
- Set the testing activities
- Clarify the dependencies for testing, what is needed in order to complete the testing
- Plan where, when, who is going to do the testing activities/testing types



1.2.3 Test Analysis and Design

Test preparation means all activities needed to be performed in order to be able to execute testing, general testing objectives are transformed into tangible test scenarios.

Why?

- Prepare all activities needed to be performed in order to be able to execute the actual testing
- Analyze what to test
- Design how to test

How?

- Review in more details the requirements or other input in order to identify what needs to be tested
- Create, review, update the tests
- Create / Identify / Update the data to test with
- Identify/set/configure the environments where to test

1.2.4 Test Execution

Why?

- Compare the requirements with the actual results of the product
- Give a feedback about the quality of the product

How?

- Run tests
- Compare the expected results with actual results
- Document all identified defects
- Analyze results



- Check the planned testing scope vs. execution
- Test environments maintenance (clean-up, restore, backup, configurations)

1.2.5 Test Reporting

Why?

- Assess the test execution results against the proposed quality measures
- Evaluate, document and communicate the product quality and readiness to be sent to the client

How?

- Evaluate the test results
- Review existing opened defects
- Provide recommendations if the product can / cannot be sent to the client
- Create test reports for stakeholders as agreed

1.2.6 Test Closure

Why?

- Clean up your “playground”
- Do a retrospective

How?

- Archive the testing documents
- Do lessons learnt meetings



1.3 QA vs. QC vs. Testing

Software Quality Assurance (SQA)

- A process that monitors and controls all software engineering processes and standards meant to ensure quality of products
- This is a prevention process
- This is applied to the entire software development process
- Assure the compliance with a predefined set of processes / activities
- In some cases ensure the conformance with standards as ISO 9000 or CMMI

Software Quality Control (QC)

- A process to ensure that a software product meets its quality objectives for the customer
- This is a defect identification process
- A set of activities designed to evaluate a product through different techniques

Testing

- Involves the execution of a software to evaluate different criteria:
 - Requirements compliance
 - Correct response for different kinds of inputs
 - Acceptable response times
 - Usable
 - As client required
 - Support on different systems/devices
- The main intent of software testing is to find defects.



2 Bug Management

2.1 What is a software bug?

- An error in an application or system which causes an incorrect result or unintended behavior
- When the expected behavior and actual behavior are not matching
- Buggy is the term used for a large number of bugs
- Also known as defect , fault, issue, error

Bug examples: the application crashes while login, unauthorized access for a certain user

2.2 Bug Reporting

Why?

- Monitor and control bugs
- Bug management

Where?

- Bug management tools; e.g Jira, Bugzilla, HP Quality Center (QC), Microsoft Team Foundation Server (TFS), Mantis
- Reference: http://en.wikipedia.org/wiki/Comparison_of_issue-tracking_systems

How?

- Bug reporting rules
- Take into account also the tool rules and limitations



2.3 Bug Reporting Rules

Summary

- To be short, clear, easy to understand. The feature can be added at the beginning.
- E.g. Address Book: Error while a contact is edited

Affects version

- The application version in which the defect was found.

Severity

- The bug impact over the end-user.
- E.g. :
 - o Blocker: blocks development and/or testing work
 - o Critical: crashes, loss of data, severe memory leak
 - o Major: major loss of function
 - o Normal: average loss of function, a workaround can be found
 - o Minor: minor loss of function or other problem where an easy workaround is present, UI issues

Priority

- The importance and order a bug should be fixed.
- E.g. P1, P2, P3, P4 or low, normal, high, urgent, immediate

Environment

- The environment on which the bug was identified
- OS and OS version: Windows 10, iOS 9.3
- alpha2, staging (e.g. the server names)



Component

- The component on which the bug was identified
- E.g. Search, Address Book, Alarms

Fix version

- The version in which the bug was fixed or the feature was implemented

Assignee

- The team member in charge to resolve / validate the feature / bug

Reporter

- The team member who reported the bug

Additional information should be added if this is required: screenshot (Skitchtool), estimations, due date

As bug description, always follows the template:

Steps to reproduce:

1. <step 1>.
2. <step 2>.
3. <step 3>.

Actual results:

<details for actual results, about what is not happening or it is supposed to happen>.

Expected results:

<details for expected results, on how the functionality should work according to requirements>.

Bug Description Example



New Alarm : Disable Repetition does not work

Steps to reproduce:

- 1. Set a new alarm with Enable Repetition. E.g. 1 minute repetition*
- 2. Press Disable Repetition and OK when the alarm is triggered.*
- 3. Wait the set repetition time.*

Actual Results:

The alarm triggers again even if the Disable Repetition was selected.

Expected results:

The alarm doesn't trigger again if the Disable Repetition is selected.

Bug Reporting Tips and Tricks:

- Use the grammar rules : no .at the end of a title (our Summary), . at the end of a sentences (Steps to reproduce, actual/expected results)
- Spell check on what you deliver
- Read the text before delivering
- Screenshot cropping
- All details for actual and expected results
- Some of the steps can be written shortly and as one
- The bug description should reflect exactly the UI controls names and labels; e.g. dropdown list, Schedule Shut Down
- The we, I , you pronoun should not be used while describing a bug
- The last step should not be the actual result
- Pay more attention to your English

Even if these are presented as good practices in bug reporting, always follow the project bug reporting rules!



3 Requirements Analysis

3.1 Requirements Fundamentals

What?

- Description of the system / software solution / product to be developed
- Describe what the software will do and how it is expected to perform

Why?

- To gather and agree what the client wants
- To know the scope of the system

How they are documented?

- Different types of artefacts: Software Requirements Specifications (SRS), Business Requirements Specifications (BRS), Software Architecture Document (SAD),
- Wireframes
- Diagrams

3.1.1 Functional Requirements

What?

- Describe the capabilities that a system/ software solution / product must have in terms of behavior and information that this system will manage. (BABOK –Requirements Book)
- Describe WHAT the system does

Examples?



- The customer should create an account and provide the following information: name and surname, electricity contract number, phone number, username and password.
- A withdrawal transaction asks the authenticated user to choose from a predefined set of possible amounts

3.1.2 Non-Functional Requirements

What?

- Describe the conditions under which a system / software solution / product must remain effective or qualities that a system must have. (BABOK –Requirements Book)
- Describe aspects that will impact or limit the system
- Describe HOW the system does
- Qualities of the system: performance, capacity, maintainability, reliability, usability, availability, scalability, compatibility

Examples?

- The page should load in maximum 400 msec
- The system should allow 50k users simultaneous
- The new version of the mobile site should be compatible with following versions of browsers: Chrome 7.0, Internet Explorer 8.0, Firefox 8.0

3.2 Requirements Clarification

“You cannot test something you don’t know or you don’t understand.”

Why?

- Avoid spending time on invalid requirements based on assumptions
- Catch requirements ambiguities as early as possible



- Requirements are not always clear, consistent, complete, or not covering every aspect of the system
- Refine the testing scope, what needs to be tested
- Identify if there are requirements that cannot be tested

How?

- Read and understand requirements
- Think how would you test each requirement
- Think how an end-user would use the system
- Breakdown of the requirements
- Ask questions to have your most important questions first

3.2.1 What questions should be asked?

Who?

- Who will do this?
- Who will receive the outputs?

How?

- How does it work?
- How many? How much?

Why?

- Why does this happen?

When?



- When will it start/end?
- When will this feature be used?

What?

- What is the end result of doing this action?
- What needs to happen next?

Where?

- Where would the user access the feature?
- Where would the results be visible?

3.3 Requirements breakdown

For a better requirements understanding and test analysis, the requirements are breakdown into more manageable parts.

A very useful tool is Xmind - <http://www.xmind.net/> :

- Used for mind mapping
- Breakdown the functionality



4 Test Analysis and Design

Test preparation means all activities needed to be performed in order to be able to execute testing, general testing objectives are transformed into tangible test scenarios.

Why?

- Prepare all activities needed to be performed in order to be able to execute the actual testing
- Analyze what to test
- Design how to test

How?

- Review in more details the requirements or other input in order to identify what needs to be tested
- Create, review, update the tests
- Create / Identify / Update the data to test with
- Identify/set/configure the environments where to test

4.1 Test Analysis

What?

- Analyze what to test

How?

- Review in more details the requirements or other input
- Understand what the system should do once built
- Identify what can be tested



- Breakdown the system functionalities
- Identify the testing types that can be applied, including the non-functional ones

4.2 Test Design

What?

- Create the test cases to test a product

How?

- Take each feature/functionality and identify in more detail what to test
- Identify the system inputs and outputs
- Select a subset of possible tests by using test design techniques
- Detail the steps and expected results for each identified test
- Good testing coverage

4.2.1 Test Design Techniques

Testing principle - “Exhaustive testing is impossible”

What?

- Select a good set of tests from the total number of possible tests for the system
- Minimum set of tests with maximum testing coverage

Examples?

- Equivalence Class Partitioning
- Boundary Value Analysis



4.2.2 Equivalence Class Partitioning

What?

- A black-box technique used to divide the application inputs in sets (partitions) that can be considered the same, having the same output
- Verify only one test from each set/partition

Why?

- Reduce the total number of tests
- Good testing coverage
- More or all tests from a set/partition is not going to find new bugs, because the system handles the same these inputs

How?

- Identify the system inputs
- Think what is the system output is for each input
- Eliminate inputs for which the system behaves the same

Example 1:

A text box accepts numeric values between 23 and 36. Which are the equivalence partitions?

Solution:

The range is [23, 36], so the partitions will be as follows:



Partition I: values $< 23 \Rightarrow$ The value is not accepted (invalid)

Partition II: 23 to 36 \Rightarrow The value is accepted (valid)

Partition III: values $> 36 \Rightarrow$ The value is not accepted (invalid)

Example 2:

To pass an exam a student needs to score 65 points. The maximum to score is 100 points.

Which are the equivalence partitions?

Solution:

The partitions/ classes are:

Class I: values $< 65 \Rightarrow$ The student doesn't pass the exam (invalid)

Class II: values between $[65, 100] \Rightarrow$ The student passes the exam (valid)

Class III: values $> 100 \Rightarrow$ The system does not accept the value (invalid)

4.2.3 Boundary Value Analysis

What?

- A black-box technique used to cover the partitions boundaries

Why?

- To verify that the system behaves correctly for all boundaries

How?

- Identify the minimum and maximum values of each partition
- A test for each boundary is selected



Example 1:

A text box accepts numeric values between 23 and 36. Identify the boundaries that needs to be covered.

Solution:

[, 22] [23 , 36] [37 ,]

Example 2:

To pass an exam a student needs to score 65 points. The maximum to score is 100 points. Identify the boundaries that needs to be covered.

Solution: [0 , 64] [65 , 100] [101 ,]

4.2.4 Test Case

What?

- A set of steps and expected results developed for a particular test

Why?

- Verify the compliance with a specific requirement
- Verify that the system works correctly

Example?

If quantity 0 the item is removed from cart

Steps:

1. Select a cart item.
2. Edit the quantity and change the value to 0.

Expected results:

The item is removed from cart. The cart total price is updated accordingly.



4.2.5 Test management tool - TestLink

What?

- Test management tool
- Used for test case management

4.3 Test Data

What?

- Specific data which is used in tests

Why?

- Verify that the system works correctly with the intended data
- Verify the system response in case of unusual, unexpected or extreme input

How?

- Data identification
- Data generation
- Production data restoration

Test Data Good Coverage:

- Valid Data
 - o Test the happy flows
 - o Real data usage
 - o Validate that the system behaves according to requirements
 - o Data is saved into database or files



- Blank / Default Data
 - o Check that correct error messages are generated
- Invalid Data
 - o Check the system behavior and that correct error messages are generated
- Volume Data
 - o Test the system with real data volume based on the application life cycle

Test data generation methods examples:

- Excel functions for numbers and dates:
 - o Rand()
 - o Randbetween()
- Browsers add-ons
 - o Chrome extension -Form Filler
- Different sites
 - o www.generatedata.com
- SQL scripts
 - o `LOAD DATA INFILE '/tmp/test.txt' INTO TABLE test`
 - o `FIELDS TERMINATED BY ',' LINES STARTING BY 'xxx'`

4.4 Test Environment

What?

- Specific locations where the tests are run
- Server-side test environment
- Client-side test environments

How?



- Installations
- Configurations

Where?

- Local machines
- Virtualization e.g. VirtualBox, VMware
- Cloud e.g. Amazon, Microsoft, Google services

Server-side test environment:

- Operating Systems
 - o Windows server versions
 - o Linux e.g. Ubuntu, Debian, Red Hat
 - o Mac OS X
- Web Servers
 - o Apache HTTP Server
 - o Apache Tomcat
 - o Microsoft Internet Information Services (IIS)
- Database
 - o Oracle
 - o MySQL

Client-side test environment:

- Operating Systems
 - o Windows
 - o Mac OS X
 - o Linux
 - o iOS



- Android
- Browsers
 - Chrome
 - Firefox
 - Internet Explorer

Test environment purpose:

- Development Environment: the environment on which development teams are working and testing the developed solution
- Test Environment: the environment where test teams are running their tests; the releases are deployed based on an agreed release process and also to address testing needs
- Staging Environment:
 - the pre-production environment, preferable similar to production system
 - the environment where clients do their validation testing
 - the staging data should mirror latest production data available
- Production Environment
 - represents the live environment, where the end-user uses the application
 - the testing is conducted here after releases to validate the deployments

Cloud Test Environment:

- Infrastructure as a Service (IaaS): physical and virtual machines, servers, storage, load balancers, network
- Platform as a Service (PaaS): database, operating systems, development tools
- Software as a Service (SaaS): email, games, communication applications



Test activities related to test environment:

- Connect to the test environment
 - o Tools: WinSCP, Putty, ssh
 - o Need to know: hostname/IP address, username and password
- Read Logs
 - o Navigate to the folder containing the log
 - o View file
 - o Usage of prompt commands (DOS, Linux)
- Installations
 - o Install new builds
 - o Install other services
- Configurations
 - o Configure database, endpoints
 - o Configure different server variables
- Continuous Integration
 - o Create builds

5 Windows/Dos command line

5.1 File system structure

Files and folders:

- Files contains data
- Folder or directory contains files
- Subfolders within the folders
- At the top is the ROOT directory



ROOT:

- The beginning of the directory tree structure
- Other usage of this term is: Root account means that by default this user has access to all commands

Disk or drive:

- Any PC partitions , e.g. c: d:

\path\

- Path of the file or folder
- Absolute path – specify the real location of a directory containing the root directory and all subfolders; e.g. c:\windows\system32
- Relative path - specify the location of a directory relative to another directory; e.g. \system32

5.2 Help commands

help

- Displays all commands and their description

<command_name> /?

- Displays the syntax and description for a command; e.g. **.dir /?** - displays the help for dir command



5.3 Directories commands

dir

- View current location content
- Syntax *DIR [disk:][\path] [/P]*
- Examples:
 - o **dir** – displays the current folder content
 - o **dir c:\windows\system32 /P** - displays the content of the system32 folder one screen at a time

cd

- Navigate from one directory to another or displays the current path location
- Syntax *CD [disk:][\path\directory_name]*
- Examples:
 - o **cd windows**
 - o **cd c:\windows\temp** - change the current directory to c:\windows\temp
 - o **cd ..** - return to the parent directory relative to the current one
 - o **cd** - return to the root
 - o **cd** - display the current path location

md

- Create new directory
- Syntax: *MD [disk:][\path\directory_name]*
- Examples:
 - o **md TEST** - create the TEST folder in the current folder
 - o **md c:\TEST\SUBTEST\NEW** - create a new folder in SUBTEST folder

rn

- Rename directory



- Syntax: *RN [disk:][\path\]directory_name*
- Examples:
 - o **rn TEST NEWNAME** - rename the TEST folder to NEWNAME
 - o **rn c:\TEST\SUBTEST\NEW OLD** - rename the NEW folder in OLD folder

rd

- Remove a directory
- Syntax: *RD [disk:][\path\]directory_name*
- Examples:
 - **rd TEST** - remove the TEST folder from current location
 - **rd c:\TEST\SUBTEST\NEW** - remove the NEW folder from the specified path

5.4 Files commands

more

- View the content of files
- Syntax: *MORE [disk:][\path\]file_name*
- **more test.txt** - displays the content of test.txt
- **more c:\TEST\SUBTEST\NEW\test.txt** - displays the content of test.txt

ren

- Rename file
- Syntax: *RN[disk:][\path\]file_name*
- Examples:
 - o **rn test.txt newfilename.txt** - rename the test.txt file to newfilename.txt
 - o **rn c:\TEST\SUBTEST\NEW\test.txt newfilename.txt** - rename the test.txt file to newfilename.txt



del

- Remove file
- Syntax: *DEL [disk:][\path\]file_name*
- Examples:
 - o **del newfilename.txt** - remove the newfilename.txt file
 - o **del c:\TEST\SUBTEST\NEW\newfilename.txt** - remove the newfilename.txt from the specified path

copy / xcopy

- Copy files or folders to a specified location
- Syntax: *XCOPY [disk:][\path\]file_name [disk:][\path\]file_name*
- Examples:
 - o **xcopy c:\TEST c:\TEST\SUBTEST\NEW** - copy the TEST folder into NEW folder
 - o **copy test.txt c:\TEST\SUBTEST\NEW\test.txt** - copy the test.txt to another location

move

- Move files or folders to a specified location [disk:][\path\]file_name
- Syntax: *MOVE [disk:][\path\]file_name*
- Examples:
 - o **move c:\TEST c:\TEST\SUBTEST\NEW** - move the TEST folder into NEW folder
 - o **move c:\TEST\SUBTEST\NEW\test.txt c:\TEST** - move the test.txt file to a new location

5.5 Other commands

cls



- Clears the prompt window
- .cls

>

- Redirect the prompt output to a file
- Examples:
 - o **dir > dir.txt** - create a new file dir.txt containing the list of folders and files from the current location
 - o **copy /? > copyhelp.txt** - create a new file copyhelp.txt containing the copy command help

exit

- Exit the command prompt
- Exampels: **exit**

5.6 Linux Commands

Here is a table with some Linux commands and their Windows correspondence.

| Description | Dos | Linux |
|-----------------------------|------------|-------|
| Displays command help | command /? | info |
| Lists directories and files | dir | ls |
| Creates a directory | mkdir | mkdir |



| | | |
|------------------------|---------------|------|
| Change directory | cd | cd |
| Check current path | chdir | pwd |
| View content of a file | more | less |
| Rename file | ren | mv |
| Copy file or directory | copy xcopy | cp |
| Delete file | del | rm |



6 Testing Types

6.1 Black-Box vs. White-box testing types

Black-box testing:

- Test the functionality of the application without knowing its internal structure, code
- Test cases are based on requirements, on what the system is supposed to do
- Testing of the output based on a particular input, without knowing how the software produces the output

White-box testing:

- Testing of internal structures and code
- Design test cases based on internal knowledge of the system next to programming skills
- Inputs are selected to check paths through the code and determine the correct outputs

Grey-box testing

- Tests covering also the structure and the functionality of the system

6.2 Static vs. Dynamic testing types

Static testing:

- Testing without running the application
- Programmers read the code to find errors –code review
- Used to find and eliminate errors or ambiguities in documents such as requirements, design, test cases

Dynamic testing



- The software is compiled and executed
- Next to functionality, the memory usage, CPU usage or other performance indicators are analyzed

6.3 Functional vs. Non-functional testing types

Functional testing:

- Test the functional requirements
- Test what the system does

Non-functional testing:

- Test the non-functional requirements
- Test how the system does
- Test the quality characteristics, like performance, data loss, security of data, recovery

6.4 Manual vs. Automated testing types

Manual testing:

- Testing is done manually without usage of tools

Automated testing:

- Usage of tools to execute tests and compare the actual results against the expected results
- Used mainly for repetitive tests
- Tools examples: Selenium suite, HP QTP (web apps), SoapUI(backend testing)



6.5 Exploratory Testing

What?

- Simultaneous learning, test design, and test execution
- Continuous tests generation based on the testers experience
- The flows in an application remain constant, it's only the choices that change, and the skill of the testers who choose the next move

Why?

- New tests are executed
- Guide the testing based on previous results
- Take less preparation
- Critical bugs are identified sooner

Disadvantages?

- The tests cannot be reviewed
- Difficult to keep track of what tests have been tested
- Hard to repeat specific details of some earlier tests

How?

- Time boxed sessions with a declarative scope
- Explore the application based on requirements and what you've learnt
- High-level tests
- Execute them
- Create new tests based on results
- Document the output of the session as you see fit
- Report the bugs



7 Testing Levels

7.1 Unit testing

What?

- A software method by which individual units of source code together with associated test data are tested to determine whether they are working or not
- This is done mainly by the development team

Why?

- Identify the issues causes as soon as possible
- For each change all code is retested

How?

- Each module/unit needs to be tested individually
- Any issue found is fixed immediately at the unit level

7.2 Component testing

What?

- Testing of software component / functionality separately by other components

Why?

- To find defects as early as possible before their integration with the other components

How?

- Each component features are tested one by one



- All tests related to the component integration with other components are not addressed during this phase

7.3 Integration testing

What?

- Testing the interactions between components only
- Usually done in the initial phase of the project when only some of the components are implemented and can be integrated
- More common the integrated components are from the same system but it is also integration testing when two components from different systems work together without having in place all the functionalities of the systems
- Components to be integrated needs to be functional as stand alones

Why?

- To validate the interactions between components are working

How?

- When more functionality is available it is highly recommended to move on to the next testing level (system or system integration testing)

7.4 System testing

What?

- Testing of all components of a single system working integrated and according to the requirements



Why?

- To detect any inconsistencies between the software units that are integrated together

How?

- Tests for all functional and non-functional aspects are executed
- System end-to-end flows are covered

7.5 System integration testing (SIT)

What?

- Testing the interactions between different (2 or more) systems

Why?

- The scope is to cover all end-to-end flows for all integrated systems

How?

- To validate the data integrity from one system to another
- All dependencies between systems are validate to function correctly

7.6 (User) Acceptance testing

What?

- It is the final testing phase before deployment
- Real life scenarios are tested by the client / end user
- Usually business requirements are validated



Why?

- Product acceptance criteria needs to be validated before going live

How?

- Tests based on acceptance criteria are run by client or end-user with the support of testers and BA
- The UAT scenarios are run on a production-like test environment
- UAT final report is created/updated with recommendation for go live/no go decision
- All involved parties agree and sign the UAT completion



8 Test Execution

8.1 Bug (fix) validation

What?

- Validate that the reported and fixed bugs are no more reproducible

How?

- Validate by priority & severity (if the priority is taken into account) or by severity
- Validate only the bugs for the current build and older
- The steps to reproduce are executed again on the fixed version, preferably on the same environment on which it was raised
- Validate the bug using various test data as inputs
- Test the bug related functionality to validate that nothing else was broken (no side effects)
- The bug is CLOSED if it does not reproduce anymore
- The bug is REOPEN if it is still reproducing or it is not fixed completely
- Provide valuable information related to the build version, test environment on which the bug validation was performed

Comments examples:

- Validated and Closed as Fixed on build 1.3.4_RC2 , iPhone 5, iOS 7.0.4
- Validated and Reopened on build 2.3_b4, podshow_3, Chrome 37.0.2062.120 mBug validation

8.2 Regression Testing

What?



- Validate that software previously developed and tested still performs correctly even after it was changed or integrated with other software

Why?

- To identify side effects due to bug fixes or other application changes
- To validate builds before deployments

How?

- Select the appropriate minimum set of tests needed to cover the change
- Execute the selected tests

8.3 Test execution good practices

Report bugs:

- Report first the major bugs first
- Let the UI and minor issues at the end

Run test cases, even regression test suite:

- Run the BAT/Smoke tests first if existing
- Run the most important features first

Exploratory testing:

- Identify the major bugs first by testing the main flows first
- Create a features importance list
- Test top-down

Bug validation



- Validate the higher severity first
- Cluster and validate the related functionality bugs





9 Test Reporting

What?

- The primary test deliverable from the test execution phase
- Assess the test execution results against the proposed quality measures

Why?

- Created for stakeholders to know the quality status of the product under test and to make further decisions

How?

- Evaluate the test results
- Review existing opened defects
- Provide recommendations if the product can / cannot be sent to the client
- Create test reports for stakeholders as agreed

TestLink test management tools reports:

- Select the project
- Under Test Reports section
- Different predefined metrics are available

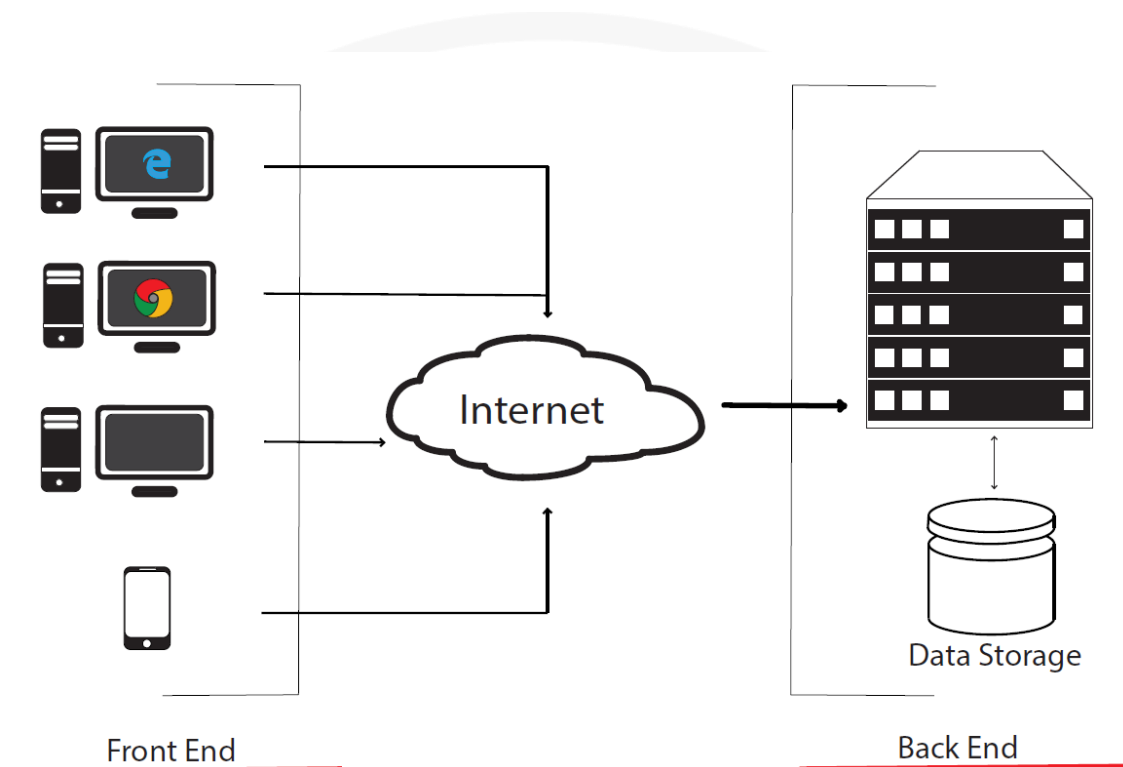
Test report documents:

- The execution overview on builds / releases
- More details are provided, including deployment recommendations



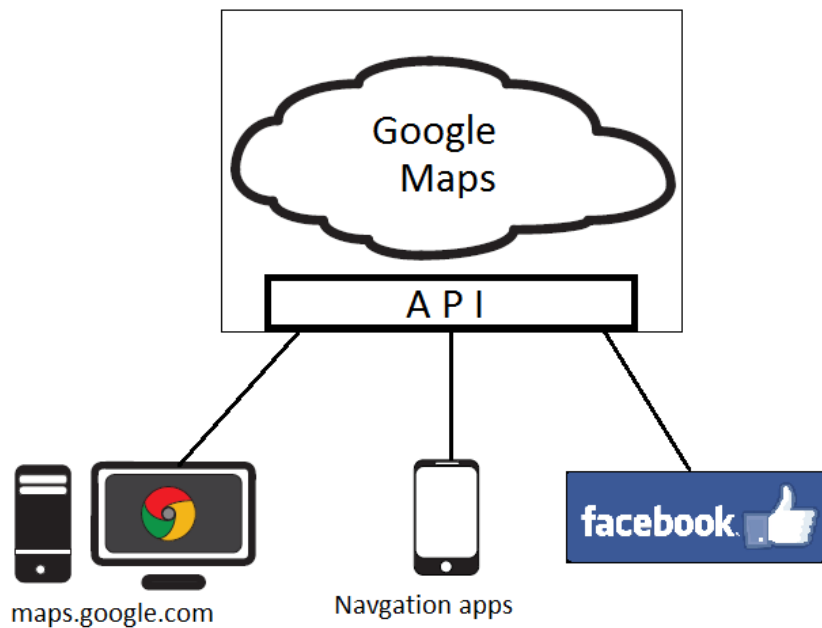
10 Testing specifics - Backend testing

10.1 Backend basics



API –Application Programming Interface

What?



- Is a set of functions and procedures for building software and applications
- Through APIs the features or data of an operating system, application, or other service are accessed and used.

Why?

- Efficiency
- Time-To-Market
- Integration
- Frontend design variation

Examples?

- Google Maps API
- YouTube APIs
- Flickr API
- Twitter APIs



10.2 Web Services

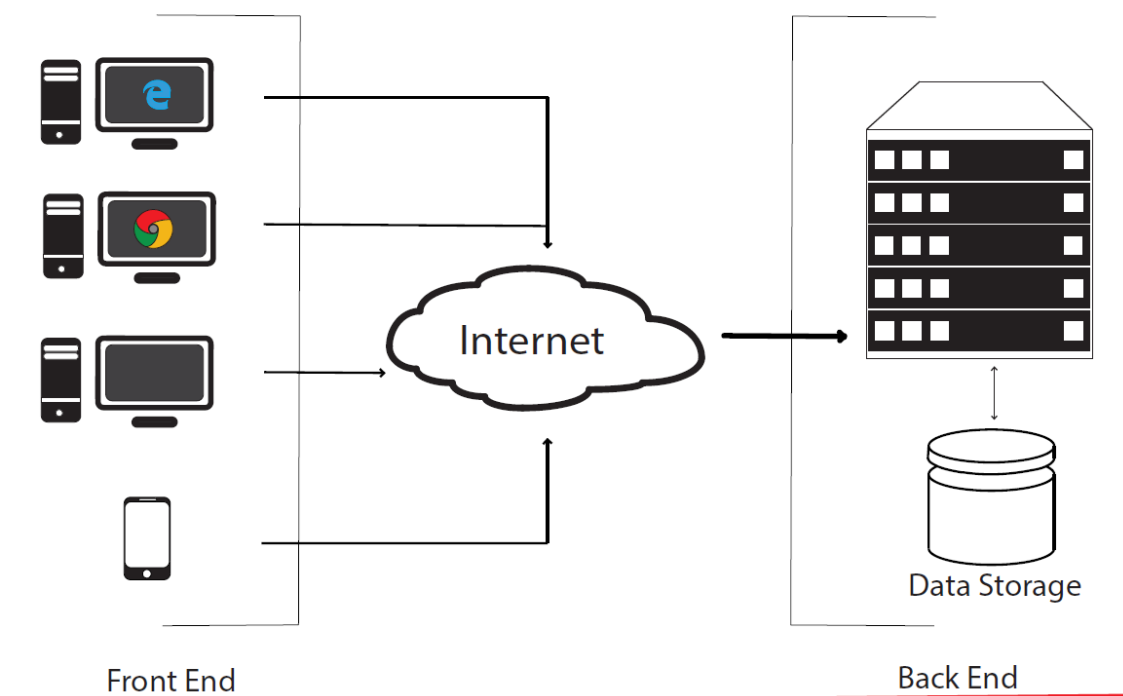
Benefits of Web Services:

- Web services are simple: it means they can be easily supported on a wide range of platforms.
- Loosely coupled: The web service may extend its interface and add new methods without affecting the clients as long as it still provides the old methods and parameters.
- Stateless: A client makes a request to a web service, the web service returns the result, and the connection is closed. There is no permanent connection. This makes it easy to scale up and out to many clients and use a server farm to serve the web services.
- Firewall-friendly: Firewalls can pose a challenge for distributed object technologies. The only thing that almost always gets through firewalls is HTTP traffic on ports 80 and 443. Because web services use HTTP, they can pass through firewalls without explicit configuration.

Disadvantage:

There is no support for bidirectional communication, which means the web server cannot call back to a client after the client disconnects.

10.3 Backend Testing



Type of Requests:

| | | | |
|--------|--------|------------------------------|---------------------------------|
| POST | Create | 201 (Created) | 404 (Not Found), 409 (Conflict) |
| GET | Read | 200 (OK) | 200 (OK), 404 (Not Found) |
| PUT | Update | 200 (OK) or 204 (No Content) | 404 (Not Found) |
| DELETE | Delete | 200 (OK) | 404 (Not Found) |

Https Statuses:



- 1xx Informational
- 2xx Success
- 3xx Redirection
- 4xx Client Error
- 5xx Server Error

Detailed list of status codes: <https://httpstatuses.com>

10.4 SOAP vs REST

| SOAP | REST |
|--|---|
| Operations (not data) E.g. switchCategory(User, OldCategory, NewCategory) | Resources (data) E.g. getUser(User) |
| Single url for operation | Different URL for each resource |
| One data format (XML only) | Different data format (XML,JSON, other) |
| Web Services Description Language | Web Application Description Language |
| Soap reads cannot be cached | REST reads can be cached |



| | |
|--|--|
| More appropriate for enterprise apps - Atomic operations | HTTP support - CRUD – Create, Read, Update, Delete |
| More secure | Better performance and scalability |

10.5 SOAP UI Tool

What?

- Tool for testing REST and Soap web services –Open Source or Pro version with extra functionalities
- Also used for performance testing
- Similar tools: jMeter

First use:

- Install Soap UI
- File .New REST Project
- Use the same urlas for the previous exercise
- Check the REST request and response
- File .New SOAP Project and Browse for a .wsdl file
- Check the SOAP request and response
- Please make sure the documentation for API / web service is available
- (Documentation for Google API .Read Documentation .Developers's guide
- <https://developers.google.com/maps/web-services/>)

Create tests in Soap UI:



- Test Suite
- Test Case
- Test Step
- Assertions
- Run Test Step / Test Case / Test Suite
- Test Logs



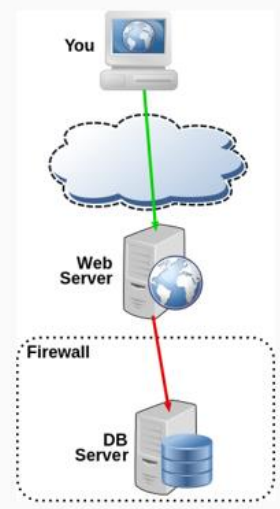


11 MySQL

11.1 Database

- Multilayer architecture
- Data access layer
- Database

Data access layer



File system

- Low Cost
- Data recovery
- No training
- No security

Database

- Control redundancy
- Security
- Multiple user access
- Integrity constraints
- Backup and recover

- Database entities

Table Name → **Products**

Column →

| ProductID | Name | Brand | Price | Quantity |
|-----------|--------------|---------|-------|----------|
| 1 | Galaxy S6 | Samsung | 3400 | 50 |
| 2 | Galaxy Alpha | Samsung | 1500 | 10 |
| 3 | iPhone 6S | Apple | 3600 | 25 |

Record →

Field →



11.2 Structured Query Language (SQL)

- Structured Query Language (SQL)
 - Declarative language for query and manipulation of relational data
- SQL consists of:
 - Data Manipulation Language (DML)
 - **SELECT, INSERT, UPDATE, DELETE**
 - Data Definition Language (DDL)
 - **CREATE, DROP, ALTER**
 - Transactional Control Language (TCL)
 - **COMMIT, ROLLBACK**

11.3 Database main operations CRUD

- CRUD for databases

| Operation | Data manipulation | Data definition |
|------------------|-------------------|-----------------|
| Create | INSERT | CREATE |
| Read (Retrieve) | SELECT | - |
| Update (Modify) | UPDATE | ALTER |
| Delete (Destroy) | DELETE | DROP |

11.4 Data Definition Language (DDL)

11.4.1 Data Types

- Numeric
 - **int** (32-bit), **bigint** (64-bit)
 - *float*, **double**
 - **decimal (scale, precision)**—for money (precise) operations



- Strings
 - **char(size)** – fixed size string
 - **varchar(size)** – variable size string
 - **text** – text data block (unlimited size)
- Binary data
 - **binary** – 0-255 binary strings
 - **varbinary** – 0-65535 binary strings
- Date and time
 - **datetime** – date and time
 - **date** – date and time (1-minute precision)
 - **timestamp** – automatically generated
- **Nullable** and **NOTNULL** types
 - All types in SQL Server may or may not allow NULL value
- **PRIMARY KEY** columns
 - Define the primary key
- **AUTO_INCREMENT** columns
 - Automatically increased values by increment when a new row is inserted (auto-increment values)
 - Used in combination with PRIMARY KEY

11.4.2 Primary Key

- Always define an additional column for PRIMARY KEY
 - Don't use an existing column (for example CNP, not safe!)
 - Must be an integer number
 - Must be declared as a PRIMARY KEY
 - Use AUTO_INCREMENT to implement auto-increment
 - Put the primary key as a first column
- Exceptions
 - Entities that have well known ID, e.g. countries (BG, DE, US) and currencies (USD, EUR, BGN)

11.4.3 Data Definition Language

- DDL commands for defining / editing objects
 - **CREATE**
CREATE DATABASE, CREATE TABLE, CREATE INDEX



- **ALTER**

ALTER TABLE

- **DROP**

DROP TABLE, DROP INDEX

11.4.3.1 CREATE

- **CREATE** command

- CREATETABLE<name>(<field_definitions>)

```
CREATE TABLE Products(  
    ProductID int AUTO_INCREMENT,  
    Name varchar(100) NOT NULL,  
    PRIMARY KEY(ProductID)  
);
```

11.4.3.2 ALTER

- **ALTER** command

- ALTER TABLE <name> <command>

```
-- Add column Country to the table Products  
ALTER TABLE Products ADD COLUMN Country int;  
  
-- Add column Country to the table Brands  
ALTER TABLE Products ADD COLUMN Country int NOT NULL;  
  
-- Remove column Country from the table Brands  
ALTER TABLE Products DROP COLUMN Country;
```

11.4.3.3 DROP

- **DROP**command

- DROP TABLE <name>



```
-- Drop table Products  
DROP TABLE Products;
```

11.5 Data Manipulation Language (DML)

11.5.1 SELECT

```
SELECT *|{[DISTINCT] column|expression [alias],...}  
FROM table WHERE clause;
```

- SELECT identifies what columns
- FROM identifies which table
- DISTINCT removes duplicates
- WHERE defines what clause

- Selecting all columns from brands

```
SELECT * FROM Products;
```

| ProductID | Name | BrandID |
|-----------|--------------|---------|
| 1 | Galaxy Alpha | 12 |
| 2 | iPhone 6S | 4 |
| 3 | HTC ONE | 273 |

- Selecting specific columns

```
SELECT  
    ProductID, Name  
FROM Products;
```

| ProductID | Name |
|-----------|--------------|
| 1 | Galaxy Alpha |
| 2 | iPhone 6S |
| 3 | HTC ONE |



11.5.2 Arithmetic Operations

- Arithmetic operators are available +, -, *, /
- Examples:

```
SELECT (7 + 3) * 2;
```

```
SELECT Name, Price, Price * 0.2 FROM Products;
```

| Name | Price | (No column name) |
|--------------|---------|------------------|
| Galaxy Alpha | 1600,00 | 384,00 |
| iPhone 6S | 3500,00 | 840,00 |
| HTC ONE | 2000,00 | 480,00 |

11.5.3 NULL Value

- A **NULL** is a value that is unavailable, unassigned, unknown, or inapplicable
 - Not the same as zero or a blank space
- Arithmetic expressions containing a **NULL** value are evaluated to **NULL**

```
SELECT Name, ProductID FROM Products;
```

| Name | ProductID |
|--------------|-----------|
| Galaxy Alpha | NULL |
| iPhone 6S | 300 |
| HTC ONE | 1 |

NULL is displayed as
empty space or as NULL

11.5.4 Column Aliases

- Aliases rename a column heading
- Useful with calculations
- Immediately follows the column name
 - There is an optional **AS** keyword



```
SELECT ProductID, Name, Price,  
Price * 0.2 AS VAT FROM Products;
```

| ProductID | Name | Price | VAT |
|-----------|--------------|---------|--------|
| 433 | Galaxy Alpha | 1600.00 | 384.00 |
| 221 | iPhone 6S | 3500.00 | 840.00 |

- Double quotation marks if contains spaces

11.5.5 Concatenation

- Concatenates columns or strings to other columns
- Is represented function CONCAT(str1, str2, ...)
- Creates a resultant column that is a character expression

```
SELECT CONCAT(Name, ' / ', Brand) AS 'Full Name',  
ProductID AS 'No.' FROM Products;
```

| Full Name | No. |
|------------------------|-----|
| Galaxy Alpha / Samsung | 134 |
| iPhone 6S / Apple | 253 |
| HTC ONE / HTC | 321 |

11.5.6 Literals

- A literal is a character, a number, or a date included in the SELECT list
- Date and character literal values must be enclosed within single quotation marks

```
SELECT Name + '''s brand is ' +  
Brand AS 'Our Products' FROM Products;
```

| Our Products |
|---------------------------------|
| Galaxy Alpha's brand is Samsung |
| iPhone 6S's brand is Apple |
| HTC ONE's brand is HTC |

11.5.7 Removing Duplicates



- The default display of queries is all rows, including duplicate rows

```
SELECT ProductID
FROM Products;
```

| ProductID |
|-----------|
| 7 |
| 7 |
| 2 |
| ... |

- Eliminate duplicate rows by using the **DISTINCT** keyword in the **SELECT** clause

```
SELECT
DISTINCT ProductID
FROM Products;
```

| ProductID |
|-----------|
| 7 |
| 2 |
| ... |

11.5.8 UNION, INTERSECT, EXCEPT

- **UNION** combines the results from several **SELECT** statements
 - The columns count and types should match

```
SELECT Name AS ProductName
FROM Products
UNION
SELECT Name AS ProductName
FROM Promotions;
```

| ProductName |
|--------------|
| Galaxy Alpha |
| iPhone 6S |
| HTC ONE |
| ... |

- **INTERSECT/ EXCEPT** perform logical intersection / difference between given two sets of records

11.5.9 Limit Selected Rows

- Restrict the rows returned by using the **WHERE** clause:

```
SELECT Name, Brand FROM Products
WHERE Brand = 'Samsung';
```

| Name | Brand |
|--------------|---------|
| Note 5 | Samsung |
| Galaxy S5 | Samsung |
| Galaxy Alpha | Samsung |
| ... | ... |



- More examples:

```
SELECT Name, Price, Description FROM Products  
WHERE ProductID = 123;
```

```
SELECT Name, Price FROM Products  
WHERE Price <= 2000;
```

11.5.10 Other Comparison

- Using **BETWEEN** operator to specify a range:

```
SELECT Name, Price FROM Products  
WHERE Price BETWEEN 1000 AND 2000;
```

- Using **IN/ NOT IN** to specify a set of values:

```
SELECT Name, Description, BrandID FROM Products WHERE BrandID IN  
(109, 3, 16);
```

- Using **LIKE** operator to specify a pattern:

```
SELECT Name FROM Products  
WHERE Name LIKE 'S%';
```

- % means 0 or more chars; _ means one char

11.5.11 Comparing with NULL

- Checking for **NULL** value:

```
SELECT Name, BrandID FROM Products  
WHERE BrandID IS NULL;
```

```
SELECT Name, BrandID FROM Products  
WHERE BrandID IS NOT NULL;
```

- Attention: **COLUMN = NULL** is always false!



```
SELECT NAME, BrandID FROM Products  
WHERE BrandID = NULL;
```

This is always false

11.5.12 Logical Operators

- Using **NOT**, **OR** and **AND** operators and brackets:

```
SELECT Name, BrandID FROM Products  
WHERE Price >= 1000 AND Brand LIKE 'S%';
```

```
SELECT Name FROM Products  
WHERE BrandID IS NOT NULL OR Name LIKE '%5_';
```

```
SELECT Name FROM Products  
WHERE NOT (ProductID = 3 OR ProductID = 4);
```

```
SELECT Name, Description FROM Products  
WHERE (ProductID = 3 OR ProductID = 4) AND  
      (Price >= 1000 OR ProductID IS NULL);
```

11.5.13 ORDER BY

- Sort rows with the **ORDERBY** clause
- **ASC**: ascending order, default
 - **DESC**: descending order



```
SELECT Name, Price  
FROM Products  
ORDER BY Price;
```

```
SELECT Name, Price  
FROM Products  
ORDER BY Price DESC;
```

| Name | Price |
|--------------|-------|
| Galaxy Alpha | 1600 |
| HTC ONE | 2000 |
| iPhone 6S | 3500 |

| Name | Price |
|--------------|-------|
| iPhone 6S | 3500 |
| HTC ONE | 2000 |
| Galaxy Alpha | 1600 |

11.5.14 INSERT

➤ INSERT command

- INSERT INTO <table> VALUES (<values>)
- INSERT INTO <table> (<columns>) VALUES (<values>)
- INSERT INTO <table> SELECT <values>

```
INSERT INTO Promotions  
VALUES (101, 'Xperia Z1', 2200);  
  
INSERT INTO Promotions(ProductID, Name, Price)  
VALUES (101, 'Xperia Z1', 2200);  
  
INSERT INTO Promotions(ProductID, Name, Price)  
SELECT ProductID, Name, Price FROM Products;
```

11.5.15 UPDATE

➤ UPDATE command

- UPDATE <table> SET <column = expression> WHERE <condition>
- Note: Don't forget the **WHERE** clause!



```
UPDATE Products SET Name = 'Galaxy A5'  
WHERE ProductID = 1;
```

```
UPDATE Products  
SET Price = Price * 1.10, Description = 'Smartphone...'  
WHERE ProductID = 3;
```

11.5.16 DELETE

- Deleting rows from a table
 - **DELETE FROM** <table> WHERE <condition>

```
DELETE FROM Products WHERE ProductID = 1;  
DELETE FROM Products WHERE Name LIKE '%5%';
```

- Note: Don't forget the **WHERE** clause!
- Delete all rows from a table at once
 - **TRUNCATE TABLE** <table>

```
TRUNCATE TABLE Users;
```

11.6 Multiple Tables

- Sometimes you need data from more than one table:





11.6.1 Cartesian Product

- This will produce Cartesian product:

```
SELECT Name, Brand FROM Products, Brands;
```

- The result:

| Name | Brand |
|--------------|---------|
| Galaxy Alpha | Samsung |
| iPhone 6S | Samsung |
| HTC ONE | Samsung |
| Galaxy Alpha | HTC |
| iPhone 6S | HTC |
| HTC ONE | HTC |
| .. | .. |

- A Cartesian product is formed when:
- A join condition is omitted
 - A join condition is invalid
 - All rows in the first table are joined to all rows in the second table
- To avoid a Cartesian product, always include a valid join condition

11.6.2 Equijoins

- Inner joins with join conditions pushed down to the **WHERE** clause



```
SELECT p.ProductID, p.Name, p.BrandID, b.BrandID, b.Brand  
FROM Products p, Brands b WHERE p.BrandID = b.BrandID;
```

| ProductID | Name | BrandID | BrandID | Brand |
|-----------|--------------|---------|---------|---------|
| 1 | Galaxy Alpha | 7 | 7 | Samsung |
| 2 | HTC ONE | 4 | 4 | HTC |
| 3 | iPhone 6S | 1 | 1 | Apple |

11.6.3 INNER JOIN

- To specify arbitrary conditions or specify columns to join, the **ON** clause is used
 - Such **JOIN** is called also **INNER JOIN**

```
SELECT p.ProductID, p.Name, p.BrandID, b.BrandID, b.Brand  
FROM Products p  
JOIN Brands b ON p.BrandID = b.BrandID;
```

| ProductID | Name | BrandID | BrandID | Brand |
|-----------|--------------|---------|---------|---------|
| 1 | Galaxy Alpha | 7 | 7 | Samsung |
| 2 | HTC ONE | 4 | 4 | HTC |
| 3 | iPhone 6S | 1 | 1 | Apple |

11.6.4 Nested SELECT

- **SELECT** can be nested in the where clause

```
SELECT Name, BrandID, Price  
FROM Products  
WHERE BrandID IN  
(SELECT BrandID FROM Brands WHERE Brand = 'HTC');
```

- Note: always prefer joins to nested **SELECT** statements for better performance



11.6.5 Nested SELECT with Aliases

- Tables from the main **SELECT** can be referred in the nested **SELECT** by aliases
- Example:
 - Find the maximal price for each Brand and the name of the product that gets it

```
SELECT Name, Price FROM Products p
WHERE Price =
  (SELECT MAX(Price) FROM Products
   WHERE BrandID = p.BrandID)
ORDER BY ProductID;
```

11.6.6 EXISTS

- Using the **EXISTS** operator in **SELECT** statements
 - Find all products with Brand from the first Brand


```
SELECT Name, Price, ProductID, BrandID
FROM Products p
WHERE EXISTS
  (SELECT ProductID
   FROM Products m
   WHERE m.ProductID = p.ProductID AND m.BrandID = 1);
```

11.6.7 Group Functions

- Group functions operate over sets of rows to give one single result (per group)



| ProductID | Price |
|-----------|----------|
| 1 | 12500,00 |
| 2 | 13500,00 |
| 3 | 43300,00 |
| 4 | 29800,00 |
| 5 | 25000,00 |
| ... | ... |



| MAX(Price) |
|------------|
| 43300,00 |

11.6.8 Group Functions

- **COUNT(*)** – count of the selected rows
- **SUM(column)** – sum of the values in given column from the selected rows
- **AVG(column)** –average of the values in given column
- **MAX(column)** – the maximal value in given column
- **MIN(column)** – the minimal value in given column

11.6.9 AVG and SUM

- You can use **AVG** and **SUM** only for numeric data types

```
SELECT
    AVG(Price) AS 'Average Price',
    MAX(Price) AS 'Max Price',
    MIN(Price) AS 'Min Price',
    SUM(Price) AS 'Price Sum'
FROM Products
WHERE Brand = 'Samsung';
```

| Average Price | Max Price | Min Price | Price Sum |
|---------------|-----------|-----------|-----------|
| 32700.00 | 32700.00 | 32700.00 | 98100.00 |



11.6.10 MIN and MAX

- You can use **MIN** and **MAX** for almost any data type (int, datetime, varchar, ...)

```
SELECT MIN(Price) MinPrice, MAX(Price) MaxPrice  
FROM Products;
```

| MinPrice | MaxPrice |
|----------|----------|
| 1600 | 3500 |

- Displaying the product's name in alphabetical order:

```
SELECT MIN(Name), MAX(Description)  
FROM Products;
```

11.6.11 COUNT

- **COUNT(*)** - returns the number of rows in the result record set

```
SELECT COUNT(*) Cnt FROM Products  
WHERE BrandID = 3;
```

| Cnt |
|-----|
| 18 |

- **COUNT(expr)** - returns the number of rows with non-null values for the expression

```
SELECT COUNT(BrandID) BrCount,  
COUNT(*) AllCount  
FROM Products  
WHERE BrandID = 16;
```

| BrCount | AllCount |
|---------|----------|
| 1 | 2 |

11.6.12 Group Function and NULL

- Group functions ignore **NULL** values in the target column

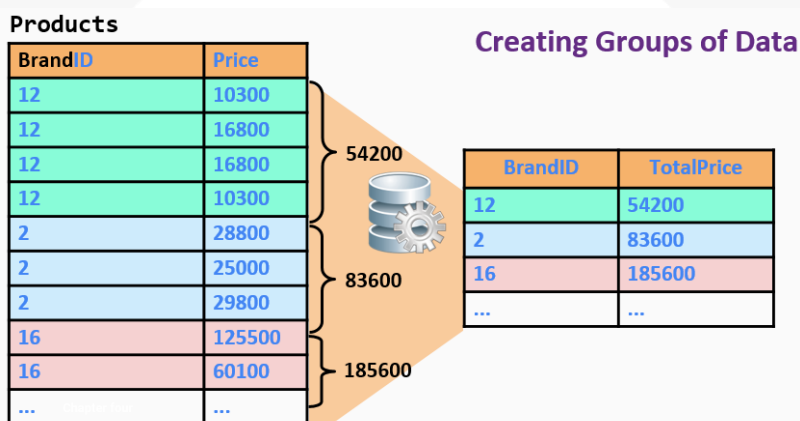


```
SELECT AVG(BrandID) Avg,  
       SUM(BrandID) / COUNT(*) AvgAll  
FROM Products;
```

| Avg | AvgAll |
|-----|--------|
| 108 | 106 |

- If each **NULL** value in the BrandID column were considered as 0 in the calculation, the result would be 106

11.6.13 Creating Groups of Data



11.6.14 GROUP BY

- We can divide rows in a table into smaller groups by using the **GROUPBY** clause
- The **SELECT+ GROUPBY** syntax:

```
SELECT <columns>, <group_function(column)>  
FROM <table>  
[WHERE <condition>]  
[GROUP BY <group_by_expression> ]  
[HAVING <filtering_expression>]  
[ORDER BY <columns>];
```

- The **<group_by_expression>** is a list of columns
- Example of grouping data:



```
SELECT BrandID, SUM(Price)
as TotalPrice
FROM Products
GROUP BY BrandID;
```

| BrandID | TotalPrice |
|---------|------------|
| 12 | 72000 |
| 2 | 108600 |
| 16 | 185600 |
| ... | ... |

- The **GROUPBY** column is not necessary needed to be in the **SELECT** list

11.6.15 Grouping by More Columns

| Brand | Product | Price | | Brand | SUM(Price) |
|---------|--------------|-------|------|---------|------------|
| Sony | Sony Xperia | 1500 | 1500 | Sony | 1500 |
| Samsung | Galaxy Alpha | 1600 | 5100 | Samsung | 5100 |
| Samsung | Galaxy S5 | 3500 | | Apple | 6500 |
| Apple | iPhone 6S | 4000 | 6500 | HTC | 3800 |
| Apple | iPhone 5S | 2500 | | LG | 2200 |
| HTC | HTC ONE | 2100 | 3800 | ... | ... |
| HTC | HTC Desire | 1700 | | | |
| LG | LG Optimus | 2200 | 2200 | | |
| ... | ... | ... | | | |

- Example of grouping data by several columns:

```
SELECT BrandID, Name,
SUM(Price) as Prices, COUNT(*) as Count
FROM Products
GROUP BY BrandID, Name;
```

| BrandID | Name | Prices | Count |
|---------|--------------|---------|-------|
| 2 | Galaxy Alpha | 58600 | 2 |
| 2 | Galaxy S5 | 50000 | 2 |
| 7 | iPhone 5S | 525000 | 21 |
| 7 | iPhone 6S | 1926000 | 157 |
| ... | ... | ... | ... |

11.6.16 HAVING

- **HAVING** works like **WHERE** but is used for the grouping functions



```
SELECT BrandID, COUNT(ProductID) as  
Count, AVG(Price) AS AveragePrice  
FROM Products  
GROUP BY BrandID  
HAVING COUNT(ProductID) BETWEEN 3 AND 5;
```

| BrandID | Count | AveragePrice |
|---------|-------|--------------|
| 2 | 4 | 27150 |
| 12 | 5 | 14400 |
| ... | ... | ... |

11.6.17 Standard Functions

- Single-row functions
 - String functions
 - Mathematical functions
 - Date functions
 - Conversion functions
- Multiple-row functions
 - Aggregate functions

11.6.17.1 String Functions

- Changing the casing – **LOWER, UPPER**
- Manipulating characters – **SUBSTRING, LENGTH, LEFT, RIGHT, LTRIM, REPLACE**

```
SELECT Name, LENGTH(Name) AS NameLength,  
UPPER(Name) AS UpperName FROM Products  
WHERE RIGHT(Name, 2) = 'Ga';
```

| Name | NameLength | UpperName |
|--------------|------------|--------------|
| Galaxy Alpha | 12 | GALAXY ALPHA |
| Galaxy S6 | 9 | GALAXY S6 |
| Galaxy Note | 11 | GALAXY NOTE |
| ... | ... | ... |

11.6.17.2 Other Functions



- Mathematical Functions – **ROUND, FLOOR, POWER, ABS, SQRT, ...**

```
SELECT FLOOR(3.14);    -- 3
SELECT ROUND(5.86, 0); -- 6.00
```

- Date Functions – **NOW, CURTIME, DAY, MONTH, YEAR, SYSDATE, ...**
- Conversion Functions – **CONVERT, CAST**

```
SELECT CAST('20151231' AS DATE);
-- 2015-12-31
```

11.7 Transactional Control Language (TCL)

11.7.1 ACID Principles

- **Atomicity**: ensures that all operations within the work unit are completed successfully; otherwise, the transaction is aborted at the point of failure and previous operations are rolled back to their former state.
- **Consistency**: ensures that the database properly changes states upon a successfully committed transaction.
- **Isolation**: enables transactions to operate independently on and transparent to each other.
- **Durability**: ensures that the result or effect of a committed transaction persists in case of a system failure.

11.7.2 Transactions

- Transactions start by executing **START TRANSACTION**
- Use **COMMIT** to confirm changes and finish the transaction
- Use **ROLLBACK** to cancel changes and abort the transaction

```
START TRANSACTION;
  DELETE FROM Products;
  DELETE FROM Brands;
  ...
COMMIT;
  ...
ROLLBACK;
```



11.7.3 Stored Procedures

- **Stored Procedures** are reusable code
- **CREATE, ALTER, DROP PROCEDURE**

```
CREATE PROCEDURE uspGetProducts  
BEGIN  
    SELECT * FROM Products;  
END
```

- Input/output parameters

```
CREATE PROCEDURE uspGetProducts  
BEGIN  
    SELECT * FROM Products;  
END
```

- Call Stored Procedures

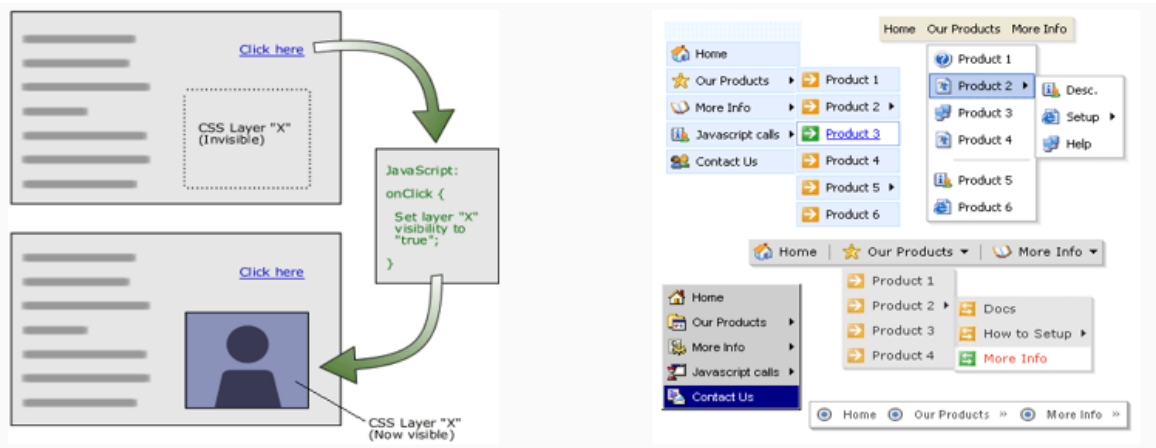
```
CALL uspGetProducts 3;  
-- OR  
CALL uspGetProducts SearchBrandID = 3;
```

- Example

```
CREATE PROCEDURE uspGetProducts SearchID INT = NULL  
BEGIN  
    SELECT * FROM Products WHERE BrandID = SearchID;  
END
```

12 JavaScript (JS)

12.1 DHTML



http://www.adobe.com/devnet/education/articles/technologies_elearning/dhtml_example.gif

<http://www.dhtml-menu.com/dhtml/apdhtml.gif>

- Dynamic HTML (DHTML)
 - Makes possible a Web page to react and change in response to the user's actions
- DHTML = HTML + CSS + JavaScript
 - DHTML
 - HTML
 - CSS
 - JavaScript
 - DOM
- HTML defines Web sites content through semantic tags (headings, paragraphs, lists, ...)
- CSS defines 'rules' or 'styles' for presenting every aspect of an HTML document
 - Font (family, size, color, weight, etc.)
 - Background (color, image, position, repeat)
 - Position and layout (of any object on the page)
- JavaScript defines dynamic behavior
 - Programming logic for interaction with the user, to handle events, etc.

12.2 Introduction to JavaScript

12.2.1 What is JavaScript?

- JavaScript is a front-end scripting language developed by Netscape for dynamic content



- Lightweight, but with limited capabilities
- Can be used as object-oriented language
- Client-side technology
 - Embedded in your HTML page
 - Interpreted by the Web browser
- Simple and flexible
- Powerful to manipulate the DOM

12.2.2 JavaScript Advantages

- JavaScript allows interactivity such as:
 - Implementing form validation
 - React to user actions, e.g. handle keys
 - Changing an image on moving mouse over it
 - Sections of a page appearing and disappearing
 - Content loading and changing dynamically
 - Performing complex calculations
 - Custom HTML controls, e.g. scrollable table
 - Implementing AJAX functionality
 -

What can JavaScript do?

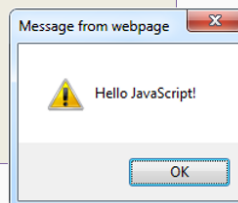
- Can handle events
- Can read and write HTML elements and modify the DOM tree
- Can validate form data
- Can access / modify browser cookies
- Can detect the user's browser and OS
- Can be used as object-oriented language
- Can handle exceptions
- Can perform asynchronous server calls (AJAX)

The First Script:

```
<html>

<body>
  <script>
    alert('Hello JavaScript!');
  </script>
</body>

</html>
```

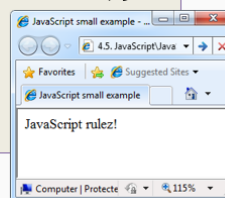


Another Small Example:

```
<html>

<body>
  <script>
    document.write('JavaScript rulez!');
  </script>
</body>

</html>
```



12.2.3 Using JavaScript Code

- The JavaScript code can be placed in:
 - **<script>** tag in the head
 - **<script>** tag in the body –not recommended
 - External files, linked via **<script>** tag the head
 - Files usually have **.js** extension

```
<script src="scripts.js">
<!-- code placed here will not be executed! -->
</script>
```

- Highly recommended
- The **.js** files get cached by the browser

12.2.4 JavaScript -When is Executed?

- JavaScript code is executed during the page loading or when the browser fires an event
 - All statements are executed at page loading
 - Some statements just define functions that can be called later
- Function calls or code can be attached as "event handlers" via tag attributes
 - Executed when the event is fired by the browser

```

```

Calling a JavaScript Function from Event Handler - Example

```
<html>
<head>
<script>
  function test(message) {
    alert(message);
  }
</script>
</head>

<body>
  
</body>
</html>
```



12.2.5 Using External Script Files

- Using external script files:

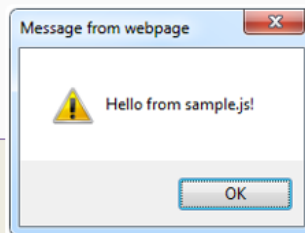
```
<img <html>
<head>
  <script src="sample.js">
  </script>
</head>
<body>
  <button onclick="sample()" value="Call JavaScript
    function from sample.js" />
</body>
</html>
```

The <script> tag is always empty

- External JavaScript file:



```
function sample() {  
    alert('Hello from sample.js!')  
}
```



12.3 JavaScript Syntax

- The JavaScript syntax is similar to C# and Java
 - Operators (+, *, =, !=, &&, ++, ...)
 - Variables (typeless)
 - Conditional statements (if, else)
 - Loops (for, while)
 - Arrays (**my_array[]**) and associative arrays (**my_array['abc']**)
 - Functions (can return value)

12.3.1 Data Types

- JavaScript data types:
 - Numbers (integer, floating-point)
 - Boolean (true / false)
- String type –string of characters

```
var myName = "You can use both single or double quotes for strings";
```

- Arrays and associative arrays (hash tables)

```
var my_array = [1, 5.3, "aaa"];  
var my_hash = {a:2, b:3, c:"text"};
```

- Undefined and null types

```
var someData; // undefined  
someData = null;
```

- Every variable can be considered as object
 - For example strings and arrays have member functions:



```
var test = "some string";  
alert(test[7]); // shows letter 'r'  
alert(test.charAt(5)); // shows letter 's'  
alert("test".charAt(1)); //shows letter 'e'  
alert("test".substring(1, 3)); //shows 'es'
```

```
var arr = [1, 3, 4];  
alert(arr.length); // shows 3  
arr.push(7); // appends 7 to end of array  
alert(arr[3]); // shows 4th element which is 7
```

12.3.2 Operations

12.3.2.1 String Operations

- The + operator joins strings

```
string1 = "fat ";  
string2 = "cats";  
alert(string1 + string2); // fat cats
```

- What is "9" + 9?

```
alert("9" + 9); // 99, + acts as concatenation
```

- Converting string to number:

```
alert(parseInt("9") + 9); // 18, + acts as addition
```

12.3.2.2 Arrays Operations and Properties

- Declaring new empty array:

```
var arr = new Array();
```

- Declaring an array holding few elements:

```
var arr = [1, 2, 3, 4, 5];
```

- Appending an element / getting the last element:

```
arr.push(3);  
var element = arr.pop();
```

- Reading the number of elements (array length):

```
arr.length;
```



- Finding element's index in the array:

```
arr.indexOf(1);
```

12.3.3 Examples

- Alert box with text and [OK] button
 - Just a message shown in a dialog box:

```
alert("Some text here");
```

- Confirmation box
 - Contains text, [OK] button and [Cancel] button:

```
confirm("Are you sure?");
```

- Prompt box
 - Contains text, input field with default value:

```
prompt("Enter amount", 10);
```

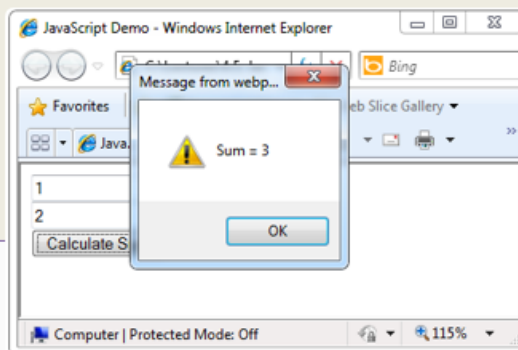
12.3.3.1 Sum of Numbers – Example

```
<html>

<head>
  <title>JavaScript Demo</title>
  <script type="text/javascript">
    function calcSum() {
      value1 = parseInt(document.mainForm.textBox1.value);
      value2 = parseInt(document.mainForm.textBox2.value);
      sum = value1 + value2;
      document.mainForm.textBoxSum.value = sum;
    }
  </script>
</head>
```

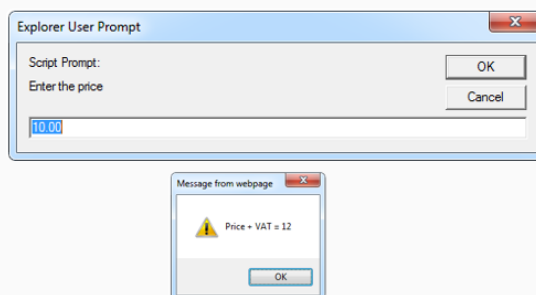
12.3.3.2 Sum of Numbers - Example (2)

```
<body>
  <form name="mainForm">
    <input type="text" name="textBox1" /> <br/>
    <input type="text" name="textBox2" /> <br/>
    <input type="button" value="Process"
      onclick="javascript: calcSum()" />
    <input type="text" name="textBoxSum"
      readonly="readonly"/>
  </form>
</body>
</html>
```



12.3.3.3 JavaScript Prompt - Example

```
price = prompt("Enter the price", "10.00");
alert('Price + VAT = ' + price * 1.24);
```



12.3.4 Statements



12.3.4.1 Conditional Statement (if)

```
unitPrice = 1.30;  
if (quantity > 100) {  
    unitPrice = 1.20;  
}  
else {  
    unitPrice = 1.40;  
}
```

| | |
|----|--------------------------|
| > | Greater than |
| < | Less than |
| >= | Greater than or equal to |
| <= | Less than or equal to |
| == | Equal |
| != | Not equal |

- The condition may be of Boolean or integer type:

```
var a = 0;  
var b = true;  
if (typeof(a)=="undefined" || typeof(b)=="undefined") {  
    document.write("Variable a or b is undefined.");  
}  
else if (!a && b) {  
    document.write("a==0; b==true;");  
} else {  
    document.write("a==" + a + "; b==" + b + ";");  
}
```

12.3.4.2 Switch Statement

- The switch statement works like in C#:

```
switch (variable) {  
    case 1:  
        // do something  
        break;  
    case 'a':  
        // do something else  
        break;  
    case 3.14:  
        // another code  
        break;  
    default:  
        // something completely different  
}
```



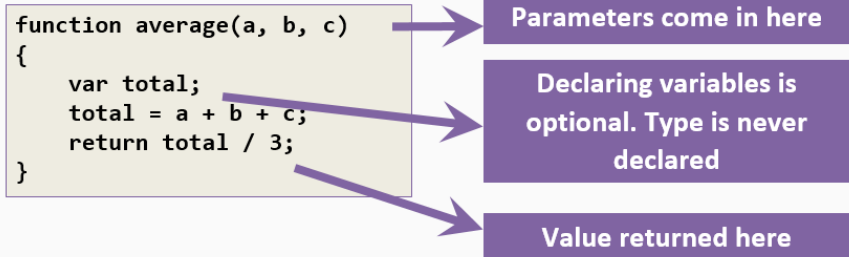
12.3.4.3 Loops

- Like in C#
 - For loop
 - While loop
 - Do...while loop

```
var counter;  
for (counter=0; counter<4; counter++) {  
    alert(counter);  
}  
while (counter < 5) {  
    alert(++counter);  
}
```

12.3.5 Functions

- Code structure –splitting code into parts
- Data comes in, processed, result returned



12.3.5.1 Function Arguments and Return Value

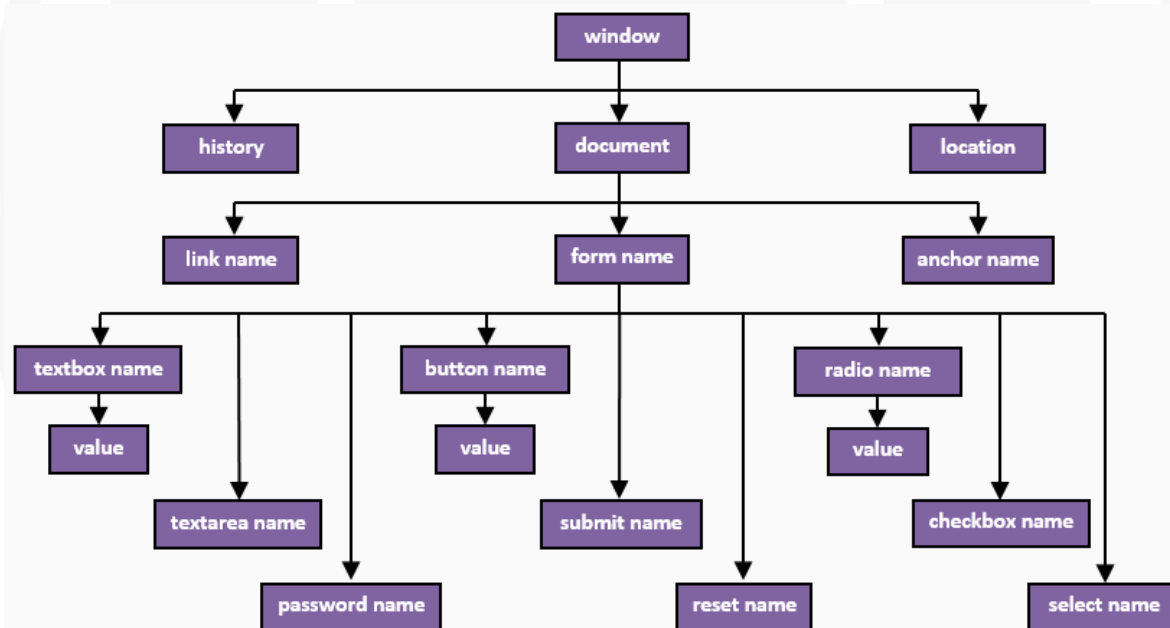
- Functions are not required to return a value
- When calling function it is not obligatory to specify all of its arguments
 - The function has access to all the arguments passed via arguments array



```
function sum() {  
    var sum = 0;  
    for (var i = 0; i < arguments.length; i ++)  
        sum += parseInt(arguments[i]);  
    return sum;  
}  
  
alert(sum(1, 2, 4));
```

12.4 Document Object Model (DOM)

Document Object Model:



- Every HTML element is accessible via the JavaScript DOM API
- Most DOM objects can be manipulated by the programmer
- The event model lets a document to react when the user does something on the page



- Advantages
 - Create interactive pages
 - Updates the objects of a page without reloading it

12.4.1 Accessing Elements

- Access elements via their ID attribute

```
var elem = document.getElementById("some_id")
```

- Via the class name attribute

```
var cls = document.getElementsByClassName("some_name")
```

- Via tag name

```
var images = document.getElementsByTagName("img")
```

- Returns array of descendant elements of the element "el"
- Instead of document, any element can be used starting access point

12.4.2 DOM Manipulation

- Once we access an element, we can read and write its attributes

```
function change(state) {  
    var lampImg = document.getElementById("lamp");  
    lampImg.src = "lamp_" + state + ".png";  
    var statusDiv = document.getElementById("statusDiv");  
    statusDiv.innerHTML = "The lamp is " + state;  
}  
...  

```

12.4.2.1 Common Element Properties

- Most of the properties are derived from the HTML attributes of the tag
 - E.g. **id**, **name**, **value**, **href**, **alt**, **title**, **src**, etc...
- Style property –allows modifying the CSS styles of the element
 - Corresponds to the inline style of the element

- Not the properties derived from embedded or external CSS rules
 - Example: **style.width**, **style.marginTop**, **style.backgroundImage**
- **className** – the class attribute of the tag
- **innerHTML**– holds all the entire HTML code inside the element
- Read-only properties with information for the current element and its state
- **tagName**, **offsetWidth**, **offsetHeight**, **scrollHeight**, **scrollTop**, **nodeType**, etc...



12.4.2.2 Accessing Elements through the DOM Tree Structure

- We can access elements in the DOM through some tree manipulation properties:
- element.childNodes
 - element.parentNode
 - element.nextSibling
 - element.previousSibling
 - element.firstChild
 - element.lastChild

```
var el = document.getElementById('div_tag');
alert(el.childNodes[0].value);
alert(el.childNodes[1].getElementsByTagName('span').id);
...
<div id="div_tag">
  <input type="text" value="test text" />
  <div>
    <span id="test">test span</span>
  </div>
</div>
```

- Warning: may not return what you expected due to Browser differences



12.4.3 The HTML DOM Event Model

- JavaScript can register event handlers
 - Events are fired by the Browser and are sent to the specified JavaScript event handler function
 - Can be set with HTML attributes:

```

```
 - Can be accessed through the DOM:

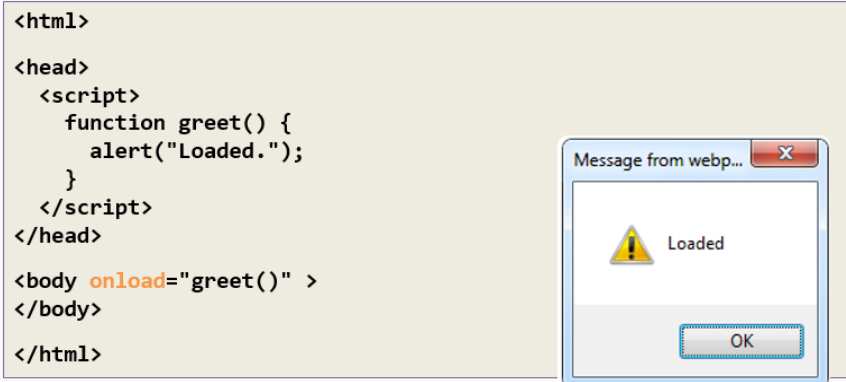
```
var img = document.getElementById("myImage");  
img.onclick = imageClicked;
```
- All event handlers receive one parameter
 - It brings information about the event
 - Contains the type of the event (mouse click, key press, etc.)
 - Data about the location where the event has been fired (e.g. mouse coordinates)
 - Holds a reference to the event sender
 - E.g. the button that was clicked

12.4.3.1 Common DOM Events

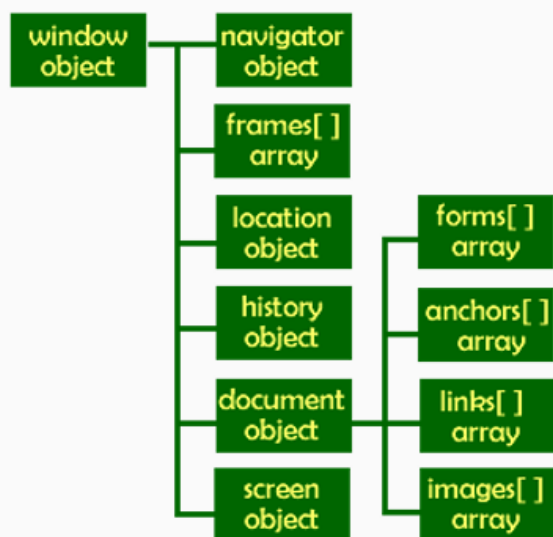
- Mouse events:
 - **onclick, onmousedown, onmouseup**
 - **onmouseover, onmouseout, onmousemove**
- Key events:
 - **onkeypress, onkeydown, onkeyup**
 - Only for input fields
- Interface events:
 - **onblur, onfocus**
 - **onscroll**
- Form events
 - **onchange** – for input fields
 - **onsubmit**
 - Allows you to cancel a form submission
 - Useful for form validation
- Miscellaneous events
 - **onload, onunload**

- Allowed only for the <body>element
- Fires when all content on the page was loaded / unloaded

12.4.3.2 onload Event - Example



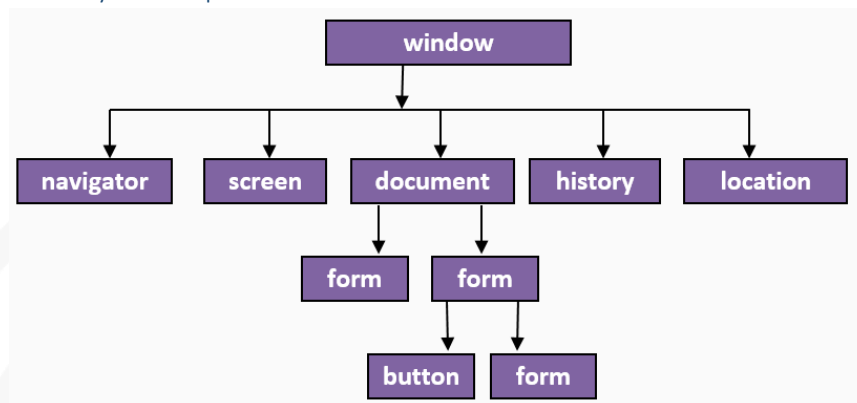
12.5 Built-in Browser Objects



- The browser provides some read-only data via:
 - window
 - The top node of the DOM tree
 - Represents the browser's window
 - document
 - holds information the current loaded document
 - screen
 - Holds the user's display properties
 - browser

- Holds information about the browser

12.5.1 DOM Hierarchy - Example

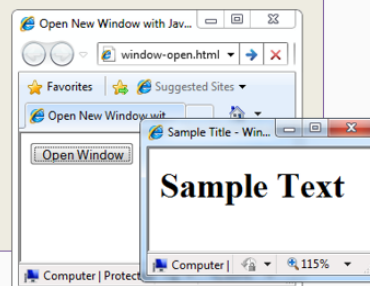


Opening New Window - Example

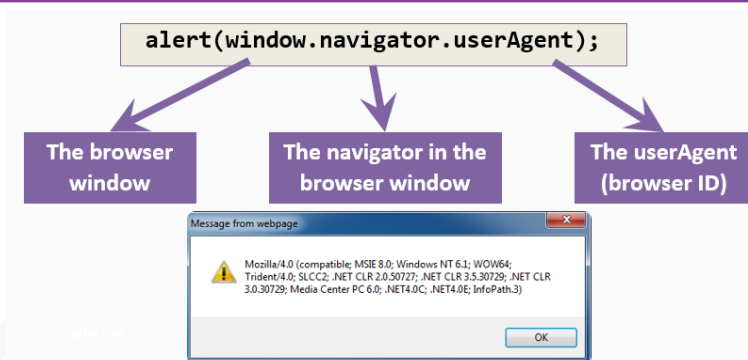
➤ `window.open()`

```
var newWindow = window.open("", "sampleWindow",  
    "width=300, height=100, menubar=yes,  
    status=yes, resizable=yes");
```

```
newWindow.document.write(  
    "<html><head><title>  
    Sample Title</title>  
</head><body><h1>Sample  
    Text</h1></body>");  
newWindow.status = "Hello folks";
```



The Navigator Object:



The Screen Object:

- The screen object contains information about the display

```
window.moveTo(0, 0);  
x = screen.availWidth;  
y = screen.availHeight;  
window.resizeTo(x, y);
```

12.5.2 Document and Location

- **documentobject**
 - Provides some built-in arrays of specific objects on the currently loaded Web page

```
document.links[0].href = "yahoo.com";  
document.write("This is some <b>bold  
text</b>");
```

- **document.location**
 - Used to access the currently open URL or redirect the browser

```
document.location = "http://www.yahoo.com/";
```

Form Validation - Example



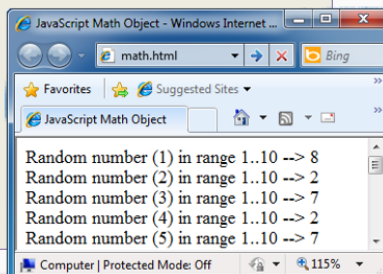
```
function checkForm()
{
    var valid = true;
    if (document.mainForm.firstName.value == "") {
        alert("Please type in your first name!");
        document.getElementById("firstNameError").style.display = "inline";
        valid = false;
    }
    return valid;
}

...
<form name="mainForm" onsubmit="return checkForm()">
    <input type="text" name="firstName" />
...
</form>
```

12.5.3 The Math Object

- The Math object provides some mathematical functions

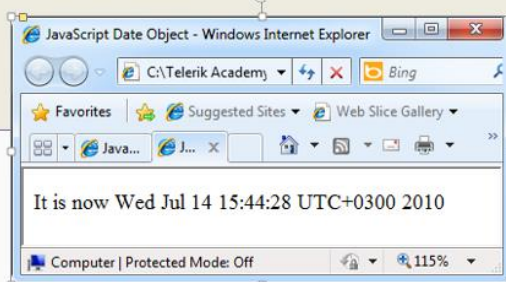
```
for (i = 1; i <= 20; i++) {
    var x = Math.random();
    x = 10 * x + 1;
    x = Math.floor(x);
    document.write(
        "Random number (" + i + ")
        in range " + "1..10 --> "
        + x + "<br/>");
}
```



12.5.4 The Date Object

- The Date object provides date / calendar functions

```
var now = new Date();  
var result = "It is now " + now;  
document.getElementById("timeField").innerText =  
    result;  
...  
<p id="timeField"></p>
```



12.5.5 Timers: setTimeout()

- Make something happen (once) after a fixed delay

```
var timer = setTimeout('bang()', 5000);
```

5 seconds after this statement executes,
this function is called

```
clearTimeout(timer);
```

Cancels this timer

12.5.6 String Functions

- Make something happen repeatedly at fixed intervals



```
var timer = setInterval('clock()', 1000);
```

This function is called
continuously per 1 second

```
clearInterval(timer);
```

Stop the timer

12.5.7 Timer - Example

```
<script type="text/javascript">
  function timerFunc() {
    var now = new Date();
    var hour = now.getHours();
    var min = now.getMinutes();
    var sec = now.getSeconds();
    document.getElementById("clock").value =
      "" + hour + ":" + min + ":" + sec;
  }
  setInterval('timerFunc()', 1000);
</script>
<input type="text" id="clock" />
```

12.6 Debugging JavaScript

- Modern browsers have JavaScript console where errors in scripts are reported
 - Errors may differ across browsers
- Several tools to debug JavaScript
 - Firebug
 - Chrome Inspector
 - Microsoft Developer Tool
- Supports breakpoints, watches
- Console editor, CSS Editor, show AJAX requests and responses



12.6.1 JavaScript Console Object

- The **console** object exists only if there is a debugging tool that supports it
 - Used to write log messages at runtime
- Methods of the **console** object:
 - **debug(message)**
 - **info(message)**
 - **log(message)**
 - **warn(message)**
 - **error(message)**





13 References

http://www.tutorialspoint.com/software_testing/

<http://sqa.stackexchange.com/>

<http://www.softwaretestinghelp.com/>

<http://www.ministryoftesting.com/>

<http://thetestingmap.org/>

<http://www.softwareqatest.com/>

<http://www.w3schools.com/sql/>