Information Management

Information Management System (IMS)

- General term for software systems designed to facilitate the storage, organization, and retrieval of information
- Sometimes used synonymously with Database Management System (DBMS)
- Data is stored in a variety of formats
- Makes it easier to access stored information

Steps when designing a database

- Requirements collection and analysis
- Conceptual design
- Choose a DBMS
- Logical Design
- Physical Design
- Implementation

IMS/DBSMS vs RDBMS

Information Management System (IMS)

Database Management System (DBMS)

- Data is generally stored in either a hierachical or navigational form

Relationship Database Management System (RDBMS)

- Data is stored in the form of tables.
- Is a special type of DBMS, which is based on a relational model

Data Model

Logical structure of a database – describes the design of the database to reflect entities, attributes, relationship among data, constraints, etc.

Types of data models:

- Object-based logical models - describes data at the conceptual and view levels (etc. ER Model)
- Record-based logical models - specify logical structure of database with records, fields, and attributes (etc. Relational Model)

Entity Relationship Model (ER Model)

Graphical approach to database design. Is a high-level data model that defines data elements and their relationships for a specified software system. An entity is something definable within a system. So entity relationship diagrams provide a visual starting point for database design - can be used to help determine information system requirements.

There are three main components in an ER diagram

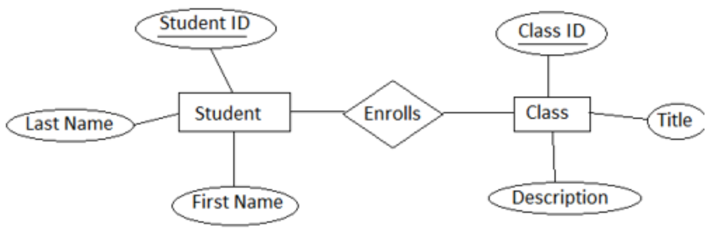
- Entity
  - Weak entity
- Attribute
  - Key
  - Composite
  - Multivalued
  - Derived
- Relationship
  - One to One
  - One to many
  - many to one
  - many to many

In an ER diagram an entity is an object represented as a rectangle. A weak entity is an entity that cannot be uniquely identified by its own attributes.

An attribute is a property of an entity - represented as an oval. Four types of attributes:

- Key - uniquely identifies an entity from an entity set. represented by an oval and the text is underlined.
- Composite - a combination of other attributes.
- Multivalued - an attribute that can hold multiple values - represented by double ovals
- Derived - value is dynamic and derived from another attribute - represented by a dashed oval.

A relationship shows the relationship among entities - represented as a diamond.



ER-to-Relational Mapping

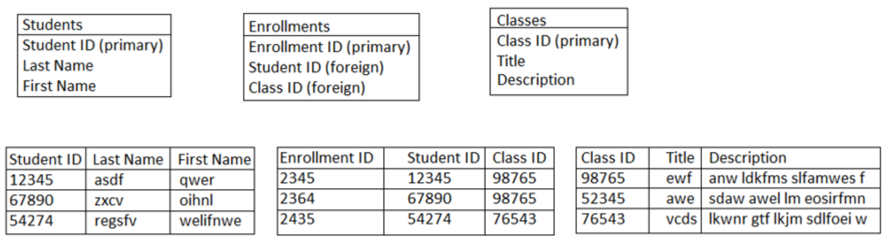
After designing the ER diagram of a system, it needs to be converted to a relational model - so it can be implemented.

Mapping entities:

- Create a table for each entity
- Entity's attributes should become fields of the table with a data type
- Declare primary key (unique identifies)

Mapping Relationships:

- Create a table for relationships
- Add the primary keys of all participating entities as a field of the table with their data types
- If a relationship has any attributes, add each attribute as a field of the table
- Declare a primary key composing all the primary keys of participating entities
- Declare all foreign key constraints



Students
Student ID (primary)
Last Name
First Name

Enrollments
Enrollment ID (primary)
Student ID (foreign)
Class ID (foreign)

Classes
Class ID (primary)
Title
Description

Student ID	Last Name	First Name
12345	asdf	qwer
67890	zxcv	oihnl
54274	regsfv	welifnwe

Enrollment ID	Student ID	Class ID
2345	12345	98765
2364	67890	98765
2435	54274	76543

Class ID	Title	Description
98765	ewf	anw ldkfms slfamwes f
52345	awe	sdaw awel lm eosirfmn
76543	vcds	lkwnr gtf lkjm sdlfoei w

Intro to SQL

Structure Query language - for relational databases. Consists of many types of statements - sometimes called sub-languages (data definition DDL and data manipulation DML).

Data definition language

- CREATE - creates new databases, tables, and view
- DROP - drops views, tables, and databases
- ALTER - modifies database schema

Data manipulation language

- SELECT - retrieve a row from a table
- INSERT - add one or more rows to a table
- UPDATE - modifies data of one or more rows from a table
- DELETE - remove one or more rows from a table

Software Testing

Purpose is to show that a program does what it is intended to do. Executed using artificial data. Can reveal the presence of errors not their absence.

Devs understand the system but will test ”gently” and is driven by delivery. On the other hand, Independent tests must learn about the system but will attempt to break it and is driven by quality.

Development testing

Testing activities carried out by the team developing the system.

Three stages:

- Unit testing - individual program unites tested. Focuses on functionality of objects or methods
- Component testing - several individual units are integrated to create components. focuses on component interfaces that provide access to the component functions
- System testing - some or all of the component are integrated and tested as a whole. focuses on component interactions.

Unit test: black-box testing

Know there is specifies function that a product has been design to perform - doesn’t know how the system works. Test demonstrate each function is fully operational while searching for errors in each function. Often performed in later stages of testing.

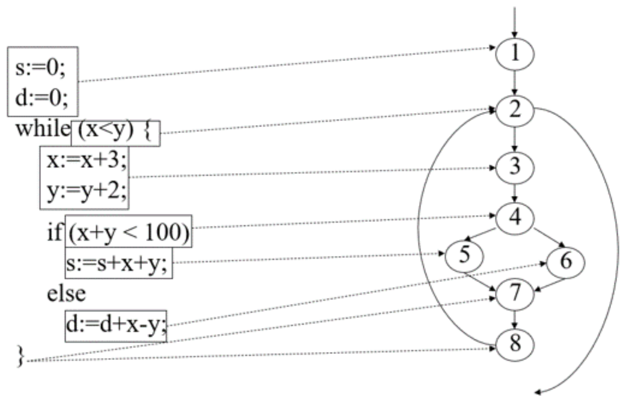
Example errors:

- incorrect or missing functions
- interface errors
- errors in data structures or external database access
- performance errors
- initialization and termination errors

White-box testing

Knows the internal workings of a product. test ensure that internal operations are performed according to specifications. Selective testing.

Control-flow-based testing. 1) from the source create a graph describing the flow of control. 2) design test cases to cover certain elements of this graph



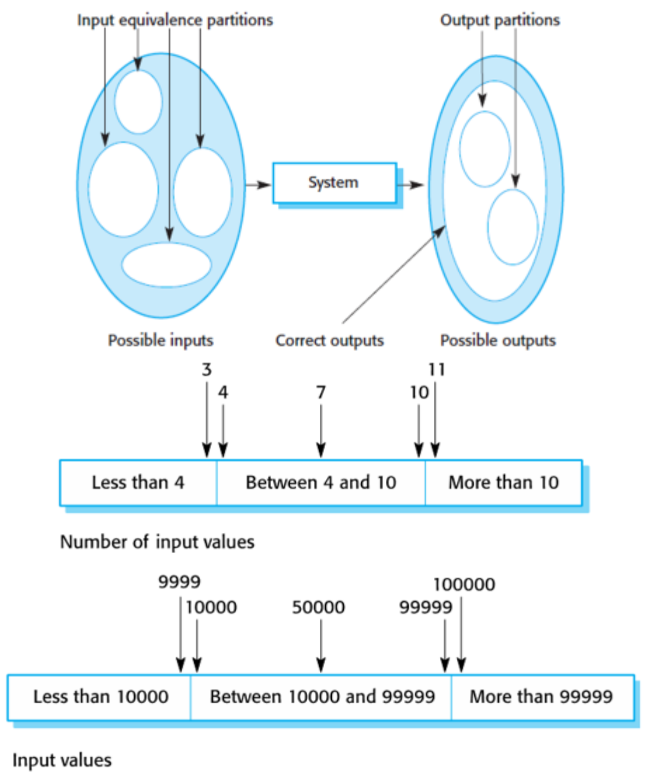
Code coverage

Metric used to verify how much of the code has been tested.

- Statement coverage - every statement in the code is executed at least once
- Branch coverage - every possible branch from a decision node is executed at least once. considered a necessary testing minimum
- Condition coverage - each boolean expression has been tested with a true false value

Partition testing

Input data and output results falls into different classes. each of these classes belong to an equivalent partition or domain. test cases should be chosen from each partition.



Boundary value analysis (BVA)

Complements equivalence partitioning. test cases on the boundary and midpoint of a partition.

Guidelines:

- if input condition range is bounded by a and b, then test cases should include a, a-1, a+1, b, b-1, b+1
- if input condition specifies a number of values, test cases should include min and max values, as well as values just above and below min/max
- If internal data structures have prescribed boundaries, test boundaries (almost full, full, overflow)

General testing guidelines

- Choose inputs that force the system to generate all error messages
- Design inputs that cause buffers to overflow
- Repeat the same input or series of inputs numerous times
- Force invalid outputs to be generated

Component Testing

Test cases are not applied to an individual component but to the interface of the composite component created by combining components

- parameter interfaces - data or function references are passed from one component to another
- shared memory interfaces - a block of memory is shared between components. data placed in memory by one component can be accessed by a different one.
- procedural interfaces - one component encapsulates a set of procedures that can be called by other components
- message passing interfaces - one component passes a message to request a service from another component. a return message includes results of executing that service.

Helps find three main types of errors:

- interface misuse - component calls another component and makes an error in use of its interface (parameters are the wrong type, passed in wrong order etc.)
- interface misunderstanding - misunderstand the specification to call a component and make an incorrect assumption about its behavior (binary search is called with an unordered array, making the search fail etc.)
- Time errors - occurs in real-time systems that use shared memory or message passing interface (out of date information etc.)

Guidelines:

- Examine code and identify each call to an external component
- Where pointers are passed, always test interfaces with null pointer parameters
- Where a component is called through a procedural interface, design tests to deliberately cause the component to fail
- perform stress testing on message passing systems
- where several components interact through shared memory, design tests to vary the order the components are activated.

System testing

testing an integrated system. checks that components are compatible, interact correctly, and transfer the right data at the right time. Overlaps with component testing

differences:

- Reusable components may have been developed separately. Component testing usually looks that the components currently being developed.
- components developed by different teams may be integrated. Sometimes system testing is performed by a team with no involvement w/ designers and devs.

Use-case testing is often used. Sequence diagrams are helpful.