Information Systems, Database Systems, and Security

Michal Valenta
michal.valenta@fit.cvut.cz

Department of Software Engineering
Czech Technical University in Prague
Faculty of Information Technology
http://fit.cvut.cz/en

June, 2010

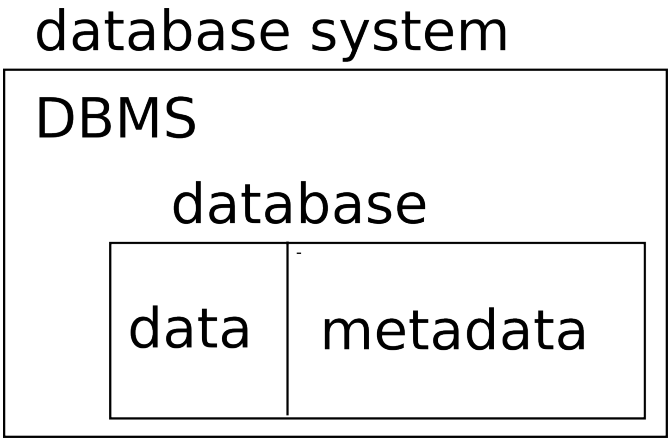
Couple of Terminology: DBS = DBMS + DB

DB Paradigm:

The data in the database exist independently on application programs.

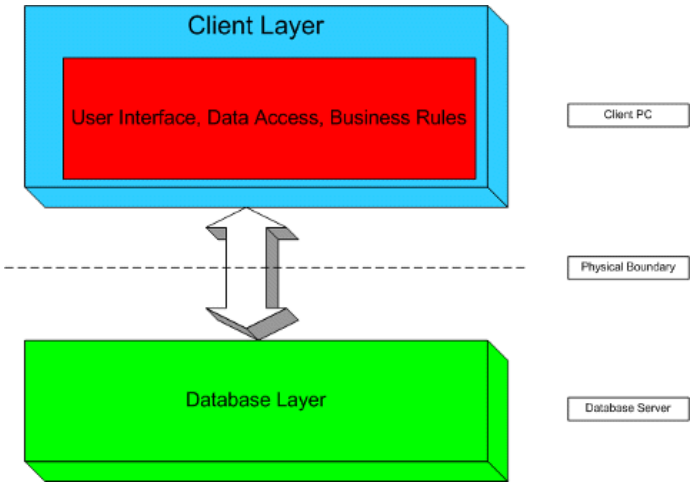
- DBS
  - DataBase System
- DB
  - DataBase
- DBMS
  - DataBase Management System

Several well-known DBMSs:
Oracle, MySQL, PostgreSQL, MS SQL , Firebird, SQL Lite

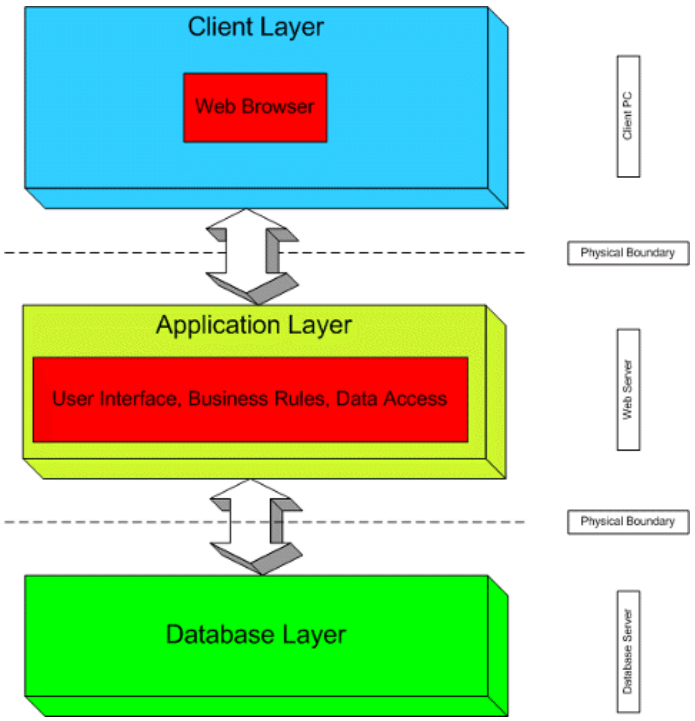


- RDBMS (Relational)
- ODBMS (Object)
- ORDBMS (Object-relational)

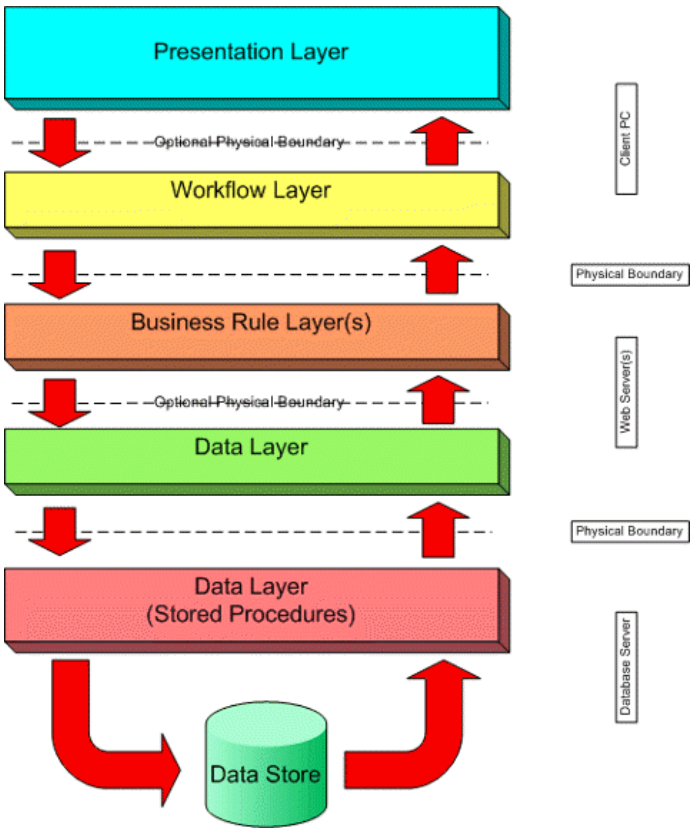
2-tier Architecture of IS



3-tier Architecture of IS



n-tier Architecture of IS



DBS in Architecture of IS – Conclusion

1. Each architecture of IS uses a data layer practically represented by a database system.
2. Data (managed by DBMS) exists independently on applications (IS presentation layer). Therefore:
   $\Rightarrow$ the same data can be accessed and changed by another IS or application (also **concurrently**),
   $\Rightarrow$ it brings both **advantage** and potential **risk of inconsistencies**.
3. DBMS must care about data from the point of view of: concurrency, consistency, persistence, and security.

History And Motivation for DBMS

- Classical data processing was based on manipulation of OS files (COBOL, PL1, Pascal).
  - 60-ties of 20th century, (50 years above)
- The authors of ISs had repeatedly implement the same tedious tasks. DBMS provides their abstract unified implementation.
- DBMS can be seen as a unified and abstract framework for data approach and manipulation. The data are encapsulated by DBMS and they are accessible exclusively through DBMS API. SQL language is an example of such API.

History and Types of DBMS

- Network DBMS, end of 60-ies, data organized into graph structure,
- Hierarchical DBMS, end of 60-ies, data organized into tree structure,
- Relational DBMS, end of 70-ies, data organized into tables, SQL language,
- Object-oriented DBMS, 80-ies, according to OO programming languages,
- Object-relational DBMS, 90-ies, the most of today DBMS, SQL incorporates OO features,
- XML-native DBMS, end of 90-ies, specialized systems (WWW, CMS, Documents).

The Main Contributions of DBMS – review

- data independence on applications,
- effective access data (query optimization),
- application building time/cost reduction,
- data integrity and protection,
- unified data backup and management,
- transactional processing,
- concurrent multiuser access,
- failure recovery.

What provides the DBMS API

- DDL – Data Definition Language,
  - allow to implement logical and physical DB scheme,
- DML – Data Manipulation Language,
- TCL – Transaction Control Language,
- DCL – Data Control Language.

How SQL implements DBMS API

SQL is designed as 4-th (programming) generation language:

1. programmers care about the result, not the algorithm,
2. it has structure of a (very simplified) natural language (English).

- DDL – CREATE, ALTER, DROP
- DML – INSERT, UPDATE, DELETE
- TCL – COMMIT, ROLLBACK
- DCL – GRANT, REVOKE, ROLE
- querying – SELECT

SQL standardization process:SQL86, SQL92, SQL99, SQL3, SQL8, ...
SQL includes others DML techniques as well (OO, XML, spatial, ...)

Several examples of SQL statements - I

Create table departments

```
CREATE TABLE departments(
  deptno serial primary key,
  dname varchar(20),
  location varchar(30));
```

Create table employees

```
CREATE TABLE emplyees (
   empno serial primary key,
   ename varchar(30) not null,
   job varchar(10) default 'clerk',
   salary number(5),
   deptno integer foreign key references departments
  );
```

Several examples of SQL statements - II

Insert a new department

```
INSERT ITNO departments (dname, location)
  values ('Marketing', 'Prague');
```

Find all analytics and return ordered list of their names

```
SELECT ename
FROM employees
WHERE job = 'analyst'
ORDER BY  ename;
```

List departments where average salary of analytics is > 6.000

```
SELECT dname, avg(salary) as analytics_avg_sal
FROM departments join employees using (deptno)
WHERE job = 'analyst'
GROUP BY deptno, dname
HAVING avg(salalary) > 6 000;
```


How to secure the database?

It is a really complex task. One should keep on mind at least following things:

1. Secure the DB connection and user/role authentication
2. HW/SW failure securing
3. Secure database against undesired access:
   1. Suitable database design
   2. Data integrity and database consistency
   3. Secure application input

1. Secure the DB connection and user/role authentication

Specific for individual DBMS. Several examples:

Oracle:

- **connection:** jdbc, odbc, SQL*Net (OCI and TCP/IP), http
- **user authentication:** database, OS, OAS, 3rd party (kerberos,...)
- in practice often stunnel to wrap SQL*Net connection

PostgreSQL:

- **connection:** jdbc, odbc, PHP, C, C++, ...
- **user authentication:** trust, reject, md5, crypt, password, krb5, ident, pam
- connection API for others clients – libPG

MySQL:

- **connection:** jdbc, odbc, PHP, .NET, C, C++, ...
- **user authentication:** password stored in database,
- connection API for others clients

2. HW/SW failure securing – what may happen

1. individual statement failure
2. transaction failure
3. connection failure
4. instance failure
5. user failure
6. HW failure

2. HW/SW failure securing – individual systems

Again, specific for individual DBMS architecture and vendor. Typically, today DBMS are capable to realize 24*7 mode.

Oracle:

- both logical and physical backup and recovery support.
- complicated, specialized tools for backup/recovery (Recovery Manager, Enterprise Manager).
- support for HA (High Available) architecture

PostgreSQL:

- backup/recovery based on pg_dump/pg_restore utilities
- support for online backup as well
- HA configuration possible

MySQL:

- backup/recovery based on mysqldump utility
- support for online backup as well
- HA configuration possible

3. Secure database against undesired access

1. Suitable database design – DDL (Views, Schemas), DCL
2. Data integrity and database consistency (Integrity constraints, transactions, TCL, DML)
3. Secure application input – beware of SQL injection

3.1. Suitable database design

- individual database accounts/schemes for individual roles
- use database VIEWS to provide accurate data projection to individual roles
- use DCL (GRANT statement) to provide access to necessary data

Screen out role clerk from seeing employee's salary

```
CREATE VIEW empview as
  SELECT empno, ename, job, depto
  FROM employees;
GRANT select ON empview TO clerk;
```

User in role clerk can't see table employees directly. He/she only can retrieve data through empview as it is a table.

3.2. Data integrity and database consistency – Integrity constraints

- Integrity constraints define the allowed (acceptable) data for the database.
- We use declarative and procedural integrity constraints in database design.
- Declarative IC: primary, unique, not null, references, check
- Procedural IC are typically more complex. For example: employee can't have greater salary than all his/her superiors.
- Most of actual DBMS support procedural extensions – triggers and stored procedures.
- Stored procedures are typically used instead of direct DML statements (in order to check complex integrity constraints).

3.2. Data integrity and database consistency – Transactional processing

- Transaction is really important feature of DBMS.
- Transaction consists of arbitrary number of DML and SELECT statements, which forms a logical unit.
- Transactions can be controlled by SQL TCL statements (BEGIN TRANSACTION, COMMIT, ROLLBACK)
- Every transaction guaranties four properties, together called ACID properties:
  - Atomicity
  - Consitency
  - Isolation
  - Durability

3.3. Secure application input – SQL injection