Software Architectures

2 SWS Lecture 1 SWS Lab Classes

Hans-Werner Sehring
Miguel Garcia

Arbeitsbereich Softwaresysteme (STS)
TU Hamburg-Harburg

HW.Sehring@tuhh.de
Miguel.Garcia@tuhh.de
http://www.sts.tu-harburg.de/teaching/ss-05/SWArch/entry.html

Summer term 2005

6. Layered Architectures & Persistence Mgmt.

1. Motivation and Fundamental Concepts
2. Revisiting Object-Oriented Analysis, Design, and Implementation
3. Design Patterns
4. Pipes & Filter Architectures
5. Event-based Architectures
6. **Layered Architectures & Persistence Management**
7. Framework Architectures
8. Component Architectures

Learning Objectives of Chapter 6

Students should be able

- to describe **layered architectures using relational DBMS as a complex example**
- to explain the benefits and liabilities of layered architectures
- to understand the layers of relational DBMS and their programming interfaces
- to understand and use the JDBC for simple tasks

Recommended Reading

- [ShGa96] Section 2.5
- [BMRSS96] Section 2.2
- Lecture Notes: Datenbanken und Informationssysteme, Arbeitsbereich STS, TUHH.
- G. Hamilton, R. Cattell, M. Fisher: JDBC Database Access with Java: A Tutorial and Annotated Reference, The Java Series, Addison Wesley, 1997.
- R. Elmasri, S.B. Navathe. Fundamentals of Database Systems. Benjamin/Cummings, Redwood City, California, 1989.

Overview

Layered Architectures

Database Management Systems (DBMS) are software systems of considerable large order. Therefore, the problems of *"Programming in the Large"* (SE problems) play a big role in their conception:

- Modularization, module specifications
- Forming groups of modules, layer construction
- Relations between modules, hierarchies

DBMS Architecture

Requirements on modules:

- Cooperative work
- Testability
- Adaptability
- Exchangeability

Other requirements for "Middleware":

Standardizability (API, ABI, network protocols)

Compatibility ("upwards" and "downwards")

Distributability (on the client / on the server)

Market strength of company (Oracle, IBM, ...)

Design Principles for Large Systems (1)

Principles:

- **Abstraction:** Concentrating on the essential, force out the irrelevant.
- **Locality:** Physical summarization of what belongs together (data and algorithms).
- **Hiding:** Restriction to the visibility of details to those parts of a system that need them.

Example: File management

- Through a*bstraction,* the set of data types and operators needed for file management is determined.
- Under the principle of *locality*, these operators are summarized in a module.
- Under the principle of *hiding,* the file management block is concealed (not needed from the outside).

Design Principles for Large Systems (2)

Principles continued:

- **Completeness:** How do you make sure, that at every abstraction level the whole intended functionality is accomplished? Often in previous system versions only the simple regular case, but not the exception is taken into consideration.
- **Verifiability:** How can you convince yourself that the particular modules – and with them also the system as a whole – fulfils the specification?
  Tests? Proofs? Special convincing efforts?

Remark: Realizable answers to these questions require an interplay between

- Programming languages,
- Design and programming methodology, and
- Programming environments.

Design Principles for Large Systems (3)

As a rule a design of *very large* systems runs out on layers of abstract machines (see next slide).

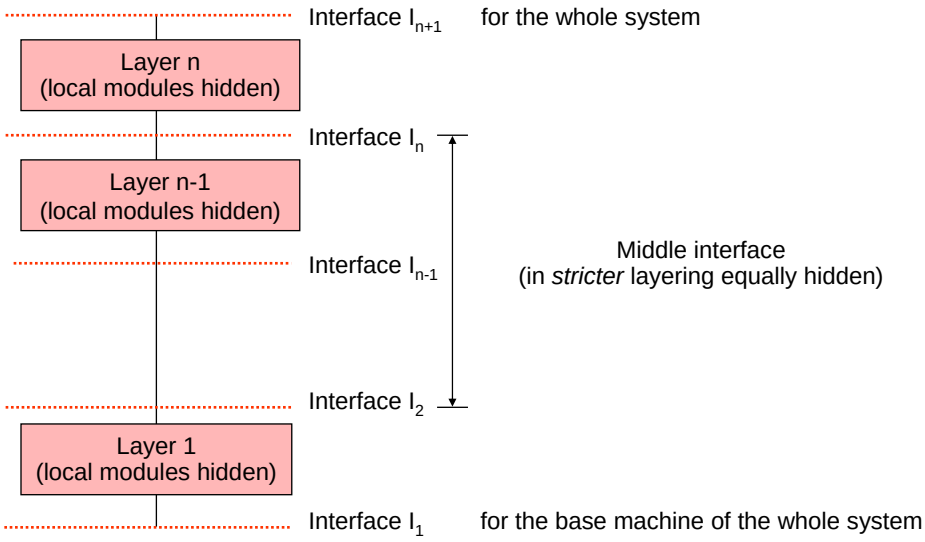
Layer construction uses the principle of abstraction to withhold the locality and hiding principles. A layer is formed often by around 10-100 modules, which can encapsulate several object types themselves.

Model of the *Abstract Machines:*

- Offers in its interface a set of data and operators (*resources*).
- Functions to work with data objects of an *abstract machine.*
- *Two* such *abstract machines* are given in each design problem:
  - the *base machine*: implemented *on it*
  - die *target machine*: *it* is implemented.

Other known examples: Operating systems (BIOS, DOS, Windows), network protocols (ISO / OSI), compiler architectures (machine code, assembler, intermediate languages, AST, ... HLL)

Design Principles for Large Systems (4)



Demands of Database Management Systems (1)

Dominating design factors:

- **Primary:** *PQRI-Demands* (Persistence, Quantity, Reactivity, Integrity)
  - Obligated database services, database services and extended database.
  - Distinguish “DBMS” Filemaker (Mac), Access (Windows)
    vs.
    Oracle (eg., Unix), IMS (IBM), …
  - Important: Scalability through layer construction
  - In recent years increasingly *distribution aspects:*
    - Application clients of DBMS server(s)
    - Replication
    - Clustering
    - Internet user to DBMS
    - ...

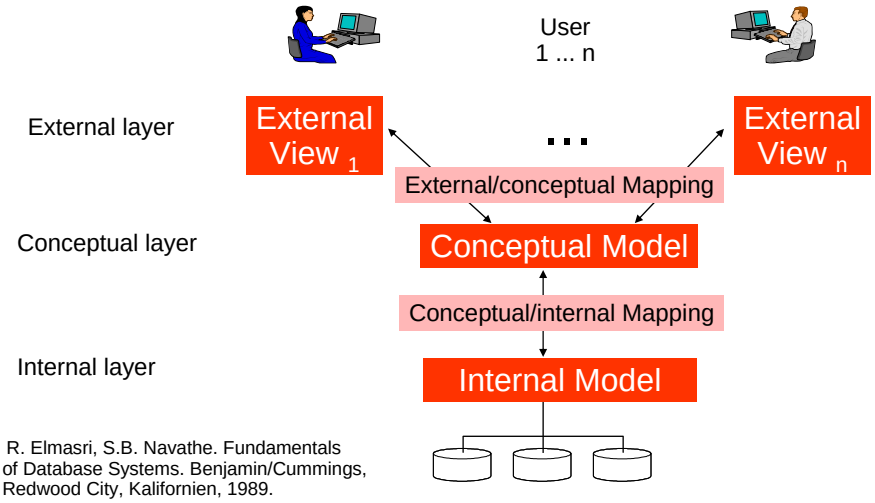
Demands of Database Management Systems (2)

- **Secondary:** *Language interfaces* (Cursor, embedded SQL, persistent objects, ...) and *semantic objects* (tree, relation, persistent heap) of the concrete data models (HDM, NDM, RDM, OODM).
- **Tertiary:** *Base machine*
  - Device interface: cylinders & blocks *(raw devices* in Unix)
  - Single user file interface (FAT, HPFS, NFS, ...)
  - Transactional file access over the operating system (Tandem, VMS)

Overview

ANSI/SPARC-Architecture (1)

The Three-layer-Architecture



see: R. Elmasri, S.B. Navathe. Fundamentals of Database Systems. Benjamin/Cummings, Redwood City, Kalifornien, 1989.

ANSI/SPARC-Architecture (2)

External layer:

- Each external model (user view: user = application program) describes the view of one or several users on the data.
- Data not relevant for the user are hidden from him.
- Example: The model of a project information system hides the salaries of the employees.

Conceptual layer:

- The conceptual model establishes the structures of the conceptual view of the whole database for the whole user community (unification of all user views in one common view).
- Entity, data type, relation and integrity conditions are taken into account.
- The physical memory structures are concealed.
- Example: The model of a company information system gathers *all* informations about an employee.

ANSI/SPARC-Architecture (3)

Internal layer:

- The internal model describes the physical memory structures of the database.
- Details of data storage and access paths are described under use of a physical data model.
- Examples:
  - Separate memory areas for permanent employees and working students.
  - B-Tree: access to projects using the project numbers

Remarks:

- The internal model is increasingly hidden (see OODBMS)
- Support for external patterns in OODBMS still insufficient
- Possibility to realize reduced relational conceptual views on NDM, HDM or file system too

ANSI/SPARC-Architecture (4)

Data independence:

- User protection in a DBMS from adverse consequences as a result of changes in the system environment.
- Types of data independence:
  - **Logical Data independence:**
    - The conceptual model can be changed without consequences for the external model.
    - Example: Expanding the database on one class or summary of several classes through generalization in the conceptual model.
  - **Physical Data independence:**
    - The internal model can be changed independent from the conceptual model, without causing functional changes in the applications.
    - Example: reorganizing the data or setting a new access path.

Overview

DBMS Resource Usage (1)

Database clients:

- As a rule (exception: “old” terminal applications) there is one operating system process per active application program (“client process").
- The database client owns the local application and DB session data (cursor, private buffer, ...) and communicates with a separate DBMS process (“server process").
- By separating the address spaces the integrity of the database is aided, and an easy possibility for remote database access is supplied. But copying of data between processes is needed.
- A client can interact with several DB servers simultaneously (eventually from different manufacturers).

Problem:

- Data exchange over the address space of the client at the element level
- Transactional processing beyond database limits

Many database manufacturers therefore offer gateways to other database systems.

DBMS Resource Usage (2)

Database server: Single Server Extension

- A single central DBMS operating system process manages numerous logical working contexts, one for each active transaction.
- In a client/server environment many clients communicate with this server, which implicitly cares for the synchronization between the local data of this client.
- The central DBMS process actually consists of a fixed number of functionally specialized, tightly coupled processes (lock manager, archive manager, application manager, ...)

Database server: Multi Server Extension

- On multiprocessor machines or in tightly coupled clusters it may be reasonable to simultaneously have several functionally equivalent DBMS operating system processes that offer access to the same database.
- The realization of multi- servers is considerably more difficult, because complex synchronization and failure recovery task have to be solved, in order to maintain the illusion of a central, consistent DBMS.

Senko's Five Layer Architecture (1)

- Strong emphasis on *data independence* (*DIAM*, *Data Independent Access Model*) (see M. Senko, 1973)
- Increased adaptation capability (new SQL Standards, new host languages, ...)
- Better portability between operating systems (Unix, VMS, MVS, OS/2, MSDOS, ...)
- Use and further development in System/R (about 1980) as well as Härder and Reuter (see database handbook)

Five layers = six interfaces!

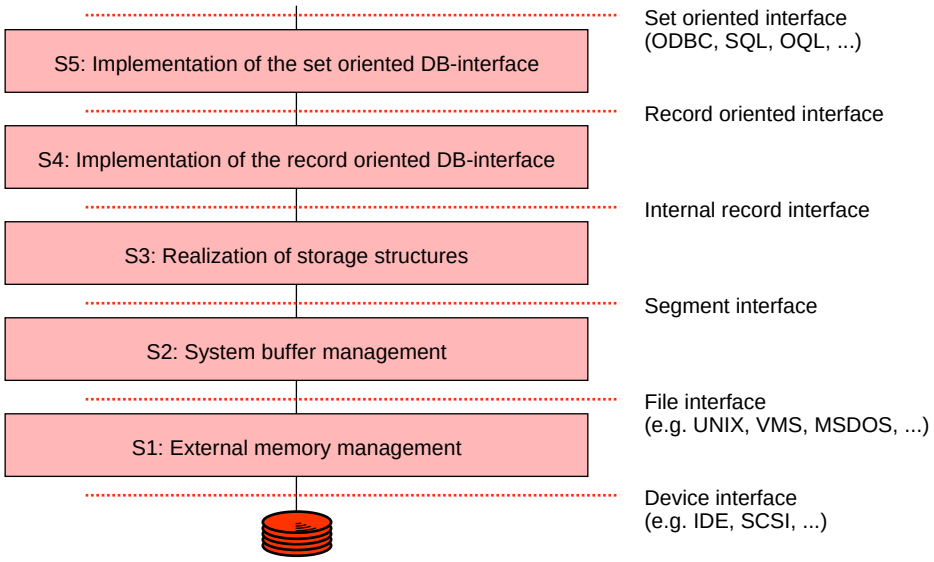
Note: The naming of layers and objects at the interfaces is fairly uneven in the references. But strong agreement on concepts.

Senko's Five Layer Architecture (2)

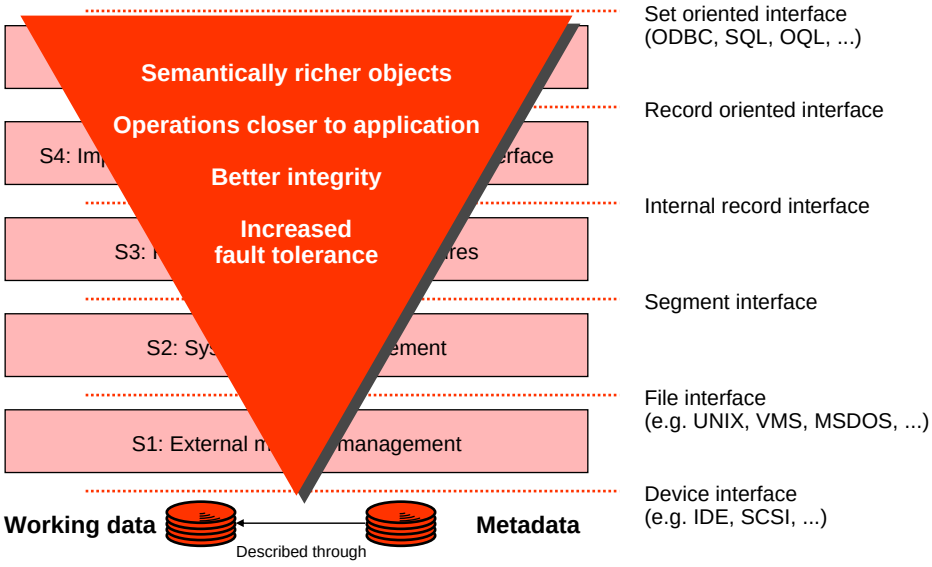
Six interfaces

- **Set oriented interface:** it supports the sets and predicates of higher database languages (SQL, OQL, DBPL) and does not apply for the network and hierarchical model.
- **Record oriented interface:** it offers logical access paths to individual records, like the navigating access in NDM and can manage transactions and control integrity demands.
- **Internal Record interface:** it has no longer reference to the concrete database model and decides over the allocation of records and record fields mainly under efficiency points of view. Access methods can be specified on it (from DBA).
- **Segment interface:** it offers a homogeneous, linear, "unending", virtual address space (in pages or segments, resp.). The access to these memory units is synchronized and can be provided in a robust manner (*error recovery*).
- **File interface:** it supplies efficient block oriented file access and is normally realized by the operating system of the host computer.
- **Device interface:** it is provided directly through hardware with adequate characteristics (direct access to persistent data).

Senko's Five Layer Architecture (3)



Senko's Five Layer Architecture (4)



S1: External Memory Management

Task and Objects:

- Realization of files consisting of blocks in physical devices
  - Simple, secure, stable writing (read/write vs. memory mapped) to data transport between system buffer and background memory
  - Direct access for working data, sequential access for protocol data

Subtasks:

- Managing physical memory on external storage media (file allocation tables, directory tables, ...) and hiding memory parameters in the process
- Block addressing: assignment of physical blocks to sub units of external storage media (cylinder, tracks)
- Inserting mirror disks and fault-tolerant hardware (RAID)
- Eventually multilevel storage hierarchy: non volatile RAM, cache in disk controller, hard disk, band robot
- Optional: dump, encryption, remote disk access, ...

S2: System Buffer Management

Tasks and Objects:

- Realization of one or more *segments* consisting of *pages* of fixed size with visible page limits
- Pages can be read and modified in slots of the *system buffers* through main memory accesses of *simultaneous* transactions
- Segments are linear address spaces that can have different characteristics (typically persistent, typically multi-user capable, typically *failure recovering*)

Motivation for *indirect* page addressing and *indirect* modification strategy

- Avoid “update in place” (immediate, direct page modification on disk)
  - Eased reversing of transactions
  - Increased parallelism with reading and writing transactions on the same page

S3: Realization of Storage Structures

Tasks and Objects:

- Preparing *physical* storage structures (see ANSI/SPARC)
  - S3a: *Record manager*: Image of the physical *record* of partially dynamic size (e.g. 100 Bytes) on pages and system buffer frame (e.g. 8K)
  - S3b: *Access path management*: Updating and use of specific physical access paths (e.g. chaining all records of a relation, primary key access and secondary key access)

The access path management is often located “above” the record manager within the layer and manages only references to records. For efficiency reasons the access path structures are themselves directly mapped to pages.

Example: B-Trees, hash tables

S4: Impl. of the Record Oriented Interface (1)

Tasks:

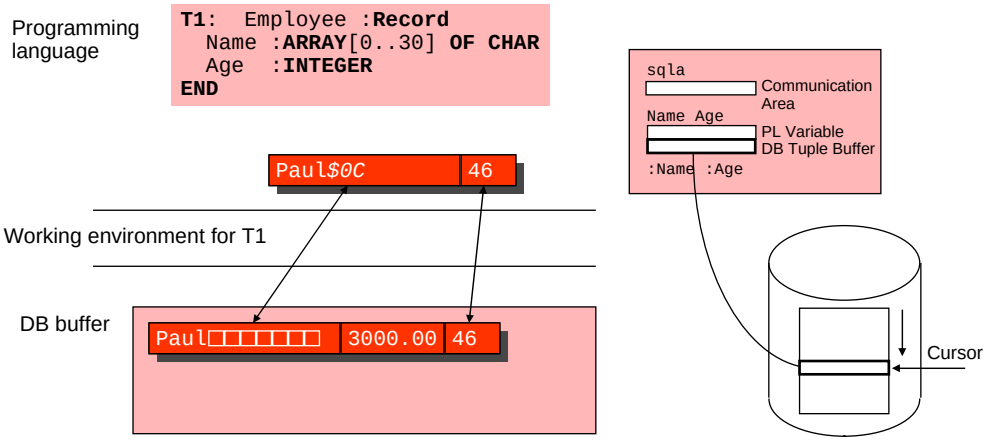
- Realization of logical sentences and logical access paths on physical records and physical access paths
- Preparing the type information for the logical records including all descriptions shown in the data dictionary
- Preparing the working environment (current state indicator, status information, controlled exception handling)
- Preparing generic functions on record attributes (sorting, middle value, extreme value, ...)
- Transaction support (BEGIN WORK, COMMIT WORK, ROLLBACK WORK) for application processes and database monitors

S4: Impl. of the Record Oriented Interface (2)

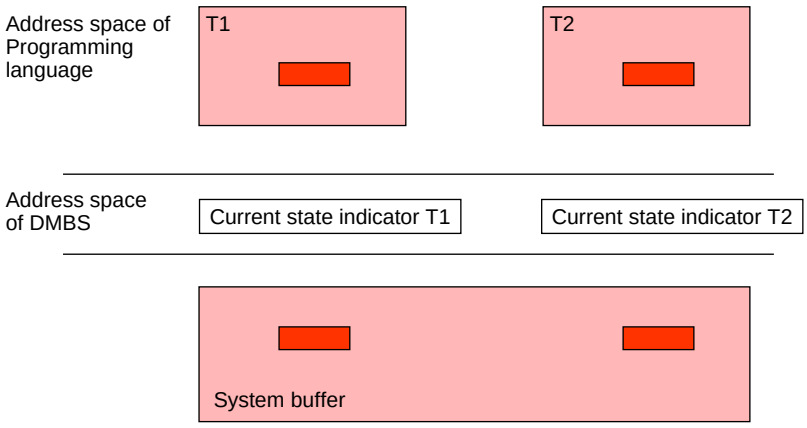
Subtasks:

- Taking care of the data dictionary, which supplies a uniformly readable and writable access to all logical and internal model information (names, owner, device assignment, access rights, modification date, ...).
- Mapping external records to internal records
  - Transformation between type mechanisms of programming languages and database
  - Reordering and simplifying attributes
  - Realizing redundant and compressed storage
  - Supporting the evolution of external types
- Maintenance of all access structures defined through the data dictionary
- Concepts for record navigation
- Session management and coordination of the (multiple layered) transaction management

S4: Impl. of the Record Oriented Interface (3)



S4: Impl. of the Record Oriented Interface (4)



S5: Impl. of the Set Oriented Interface (1)

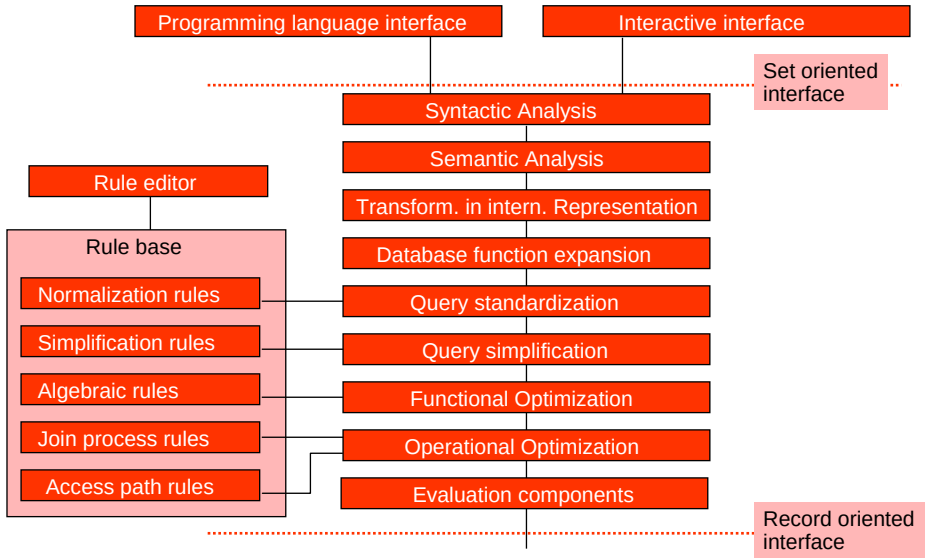
Tasks and Objects (only available in RDM and OODM):

- Mapping set oriented queries to sequences of calls of the record oriented interface
- Management of set valued query results

Subtasks:

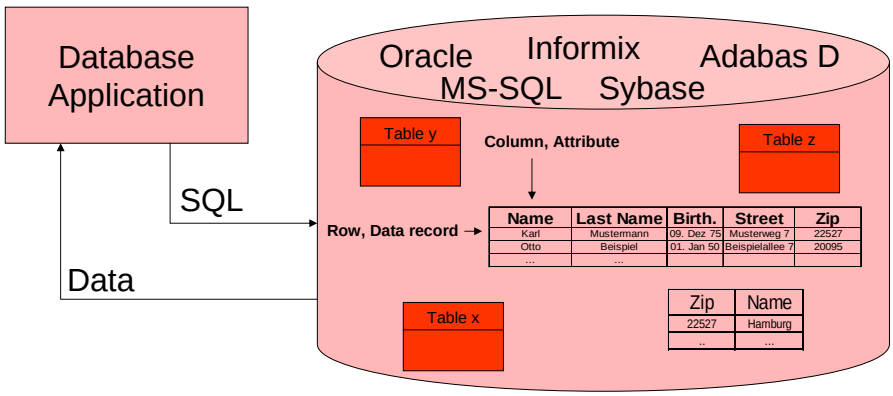
- Eventually testing declarative user defined integrity conditions
- Eventually iterative computation of recursive queries
- Query analysis
  - Relationships to programming language variables
  - Dissolving and relating the external with internal object names
  - Eliminating views

S5: Overview Query Optimization and Evaluation



Overview

Access to Relational Databases from Applications



JDBC

Java Database Connectivity

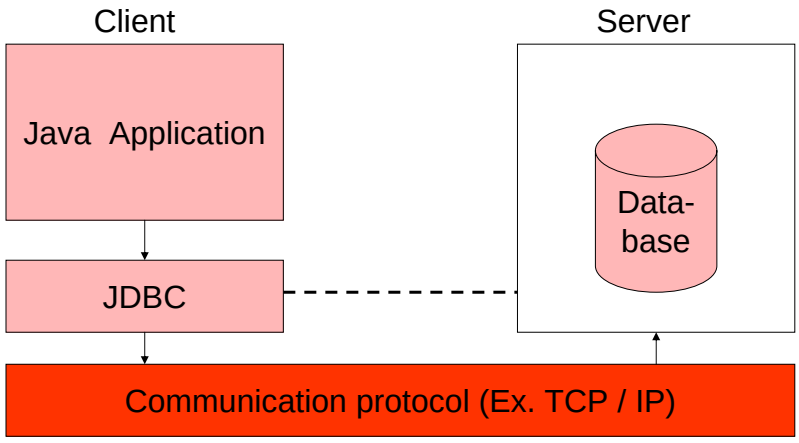
Class library for access to SQL databases; package `java.sql`

SQL commands are sent to a database as a string with help of JDBC methods

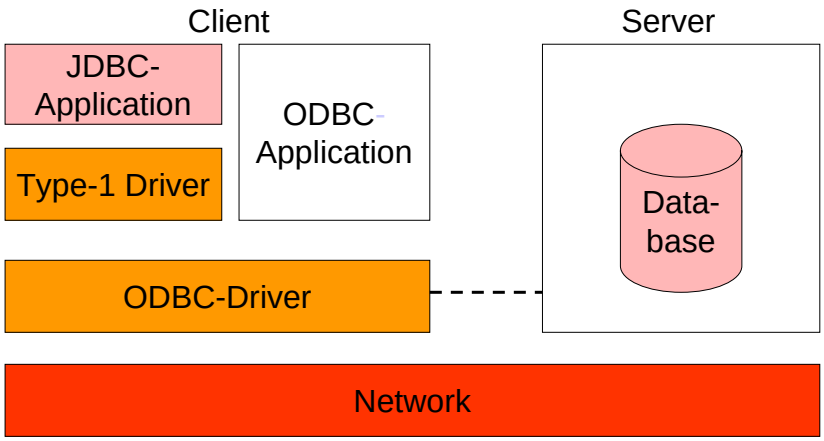
The database performs the SQL commands and delivers the requested data and success or failure messages

Data delivered from the database is read again through JDBC methods

Diagram of a JDBC Application



Type-1 Driver



Type-1 Driver (JDBC-ODBC Bridge)

Used for the communication with the ODBC-Driver available to the database

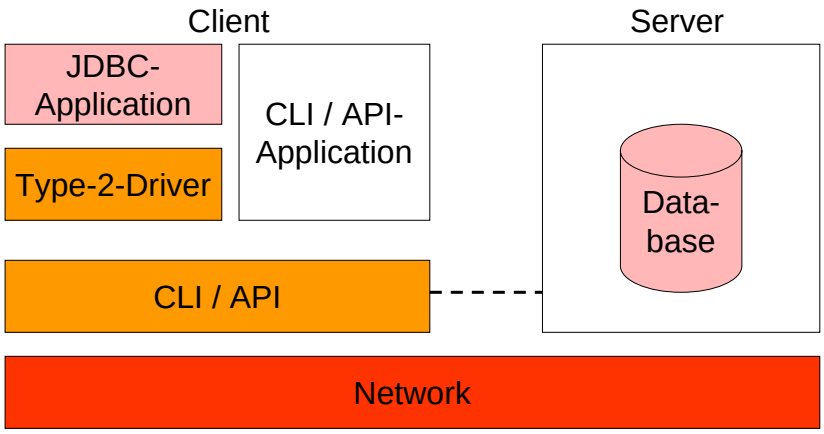
Advantage:

- Same Type-1 JDBC driver can communicate with every database system, for which a ODBC driver is available

Disadvantages:

- The ODBC driver used by the JDBC driver depends
  - on the operating system on which the database application runs and
  - on the database system used
- If the same JDBC application is deployed on another operating system, a suitable ODBC driver for this operating system is needed.
- Given that ODBC is a standard established by Microsoft, ODBC drivers are mainly available for Microsoft operating systems.

Type-2 Driver



Type-2 Driver (native API partly-Java)

Access to communication with the database on a programming interface provided by the database manufacturer (e.g., Oracle Call Interface, Informix-CLI)

Advantage:

- the JDBC Driver can be adapted by the database manufacturer to the specific capabilities of the database

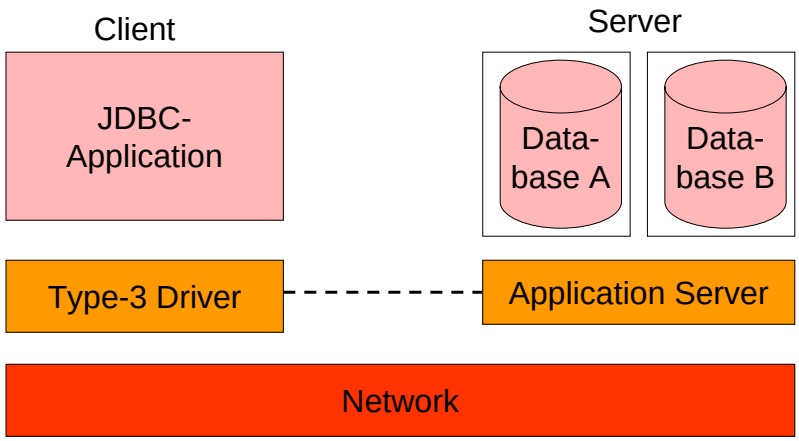
Disadvantage:

- Type-2 JDBC drivers can only communicate with the database system, for which it was programmed

Disadvantages with Type-1 and Type-2 Drivers

- The JDBC driver is only partially programmed in Java.
- The JDBC method calls must be converted into calls of another programming interface, before they can be sent to the database system.
- The implementation of this additional programming interface depends on the operating system of the client computer and on the database system used in the server and must be exchanged when changing platforms.

Type-3 Driver



Type-3 Driver (net-protocol all-Java)

- The JDBC driver does not communicate directly with a database system, but with a so called application server over a database-independent protocol
- The application server translates the messages of the client into messages that can be processed by a specific database system
- The application server can run either on the client computer, the server computer or on a third computer (Three Tier Architecture)

Type-3 Drivers

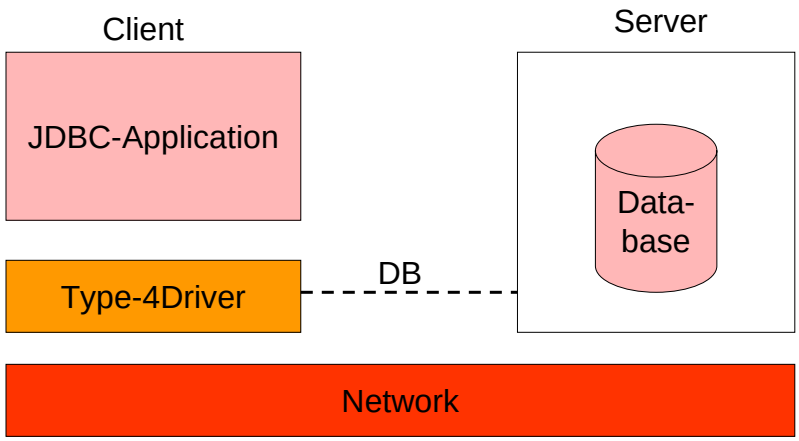
Advantages

- The driver is programmed 100% in Java and therefore works with every operating system, on which Java programs can be run.
- The driver uses a database-independent communication protocol and can therefore be used together with every database system, supported by a given application server.
- Maximum flexibility and platform-independence: if the operating system of the client computer or the database system on the server are changed, no changes to the JDBC driver of the client are needed.

Disadvantages

- The driver works together exclusively with application server of the particular middleware manufacturer.
- Only database systems supported by the application server can be used.

Type-4 Driver



Type-4 Driver (Native Protocol all-Java)

The driver is 100% programmed in Java and therefore works with every operating system, on which Java programs can be run.

The driver communicates direct with the database system over the communication protocol of the database manufacturer.

Advantages:

- No operating system-dependent ODBC driver or CLIs have to be installed on the client.
- Performance losses caused by the conversion of JDBC calls to other programming interfaces (see Types 1/2) or from one communication protocol to another (see Type 3) are avoided.

Disadvantages

- The driver can only communicate with the database system, for which it was developed.
- When changing to another database system a new driver is needed.
- Not all database manufacturers offer Type-4 drivers .

Portability of Applications

- Modern application programs, especially Java programs, should be portable.
- Sun marketing slogan: "Write once, run anywhere“ (e.g., Solaris, Windows NT, AIX, HPUX, Mac OS, Linux, …)
- These contain no computer or operating system-specific code.
- JDBC supplies a unified interface.
- SQL is a standardized query language for databases.

But:

- Not every database system supports all JDBC functions.
  - Methods
  - Isolation levels
  - …
- Subtle differences in implementation of JDBC functions.
- Database manufactures expand “their" SQL with further functions and commands, that cannot be used any more on other database systems, in case of migration.

Portability of Database Applications

Porting **database applications** to **other database** systems:

- Theoretically at most the exchange of JDBC drivers is necessary (1 File).
- In practice it must be checked, even manually, if the new database system “understands“ the dialect that the application program uses.
- If no "portable SQL" was used, the parts of the application program, in which SQL commands are sent, must be adapted to the new database system.
- Concentrate the SQL instructions and SQL generation in a few, central classes, if possible.
- Avoid SQL language constructs, that can only be processed by a specific Database system.
- Avoid database-specific SQL data types in table definition.
- Use SQL data types, for which in other database systems, either the same or at least an equivalent type exists.