Part 1: Introduction

References:

- Elmasri/Navathe:Fundamentals of Database Systems, 3rd Edition, 1999.
  Chapter 1, "Databases and Database Users"
  Chapter 2, "Database System Concepts and Architecture"
- Kemper/Eickler: Datenbanksysteme (in German), 3rd Edition, 1999.
  Chapter 1: "Einleitung und Übersicht" ("Introduction and Overview")
- Lipeck: Skript zur Vorlesung Datenbanksysteme (in German), Univ. Hannover, 1996.
- Silberschatz/Korth/Sudarshan: Database System Concepts, 3rd Edition.
  Chapter 1: "Introduction".

Exercise

Name some typical applications of databases, e.g. companies or organizations which use them, or any private uses you might have for a database:

1. ____________________
2. ____________________
3. ____________________
4. ____________________
5. ____________________

Objectives

- Database Management System Functions.
- Basic DBMS vocabulary: Schema, Instance/State, Data Model, Query, Update, DDL, DML.
- Data Independence, Three Schema Architecture, Declarativity.
- ACID transaction properties.

Overview

1. Persistence, Schema & State
2. Data Independence,
   Three-Schema Architecture
3. Other DBMS Functions, e.g. Transactions
4. Database Users and Database Tools
5. Summary

Persistent Storage (1)

Today:

5 → factorial → 120

Tomorrow:

5 → factorial → 120

⇒ No persistent storage necessary.
The output is a function of the input only.

Persistent Storage (2)

Today:

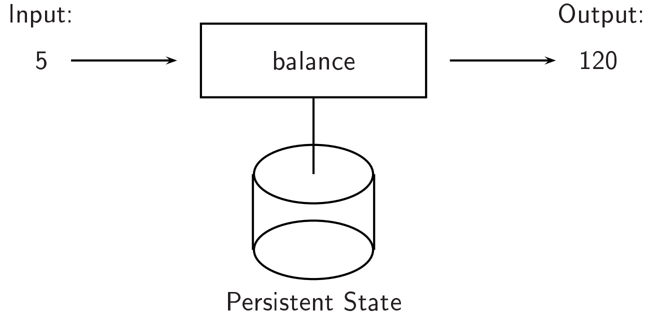
5 → balance → 120

Tomorrow:

5 → balance → 80

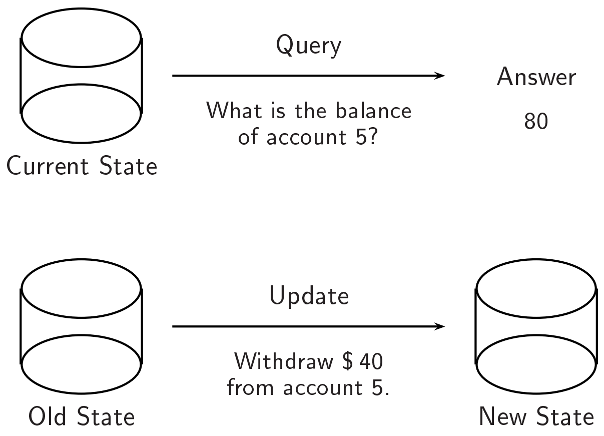
⇒ Output is a function of input and persistent state.

Persistent Storage (3)



Persistent Information: Information which lives longer than a single process / program execution.

Persistent Storage (4)



Persistent Storage (5)

Exercise:

Which of the following programs need persistent storage? Why?

- Pocket Calculator
- Web Browser
- Screen Saver
- Computer Game

  Once you've won, it starts in a higher level the next time.

Is information stored in the Windows Registry persistent?

Typed Persistent Information (1)

Classical Way to Implement Persistence:

- Information needed in other program invocations is saved in a file.
- The operating system (OS) stores the file on a disk.
- Disk is persistent memory: The contents is not lost if the computer is switched off or the operating system is rebooted.
- File systems are predecessors of modern database management systems.

Typed Persistent Information (2)

Implementing Persistence with Files:

- In most operating systems, files are sequences of bytes.
- A record structure must be defined as in Assembler languages.

0:				4:						40:				45:
0	0	0	5	S	m	i	t	h	...	0	0	1	2	0

- File structure information is contained only in the programmers' heads.
- System cannot prevent errors because it does not know the file structure.

Typed Persistent Information (3)

Implementing Persistence with a Database Management System:

- The structure of the information to be stored must be defined in a formal language which the system understands.

```
CREATE TABLE ACCOUNT(No      NUMERIC(6),
                     Owner   VARCHAR(40),
                     Balance NUMERIC(10,2))
```

- Thus, the file structure is formally documented.
- The system can prevent type errors in application programs.
- The programming is simpler (higher abstraction level).

Basic Notions (1)

Database Schema:

- Formal definition of the structure of the database contents.
- Determines the possible database states.
- Defined only once (when the database is created).
- Corresponds to variable declaration (type information).

Instance of the Schema:

- Database state conforming to the schema.
- Changes often (whenever database-information is updated).
- Corresponds to current contents/value of a variable.

Basic Notions (2)

ACCOUNT		
No	Owner	Balance
1	Ullman, J.D.	200.00
2	Codd, E.F.	150.00
3	Date, C.J.	300.00
4	Stonebraker, M.	495.00
5	Elmasri, R.	120.00
6	Navathe, S.B.	80.00

DB Schema: ACCOUNT, No, Owner, Balance

DB Instance (State): rows 1–6

Basic Notions (3)

Data Model:

- Defines a formal language (syntax & semantics) for
  - declaring database schemas (Data Definition Language),
  - querying the current database state (Query Language),
  - changing the database state.
- Often, the term “data model” is used in a more abstract way for the general concepts underlying the concrete language.
- For instance, one speaks of the “relational data model”, and not of some specific version of the SQL language.
- Unfortunately, the term “data model” is then very fuzzy.

Basic Notions (4)

Examples of Data Models:

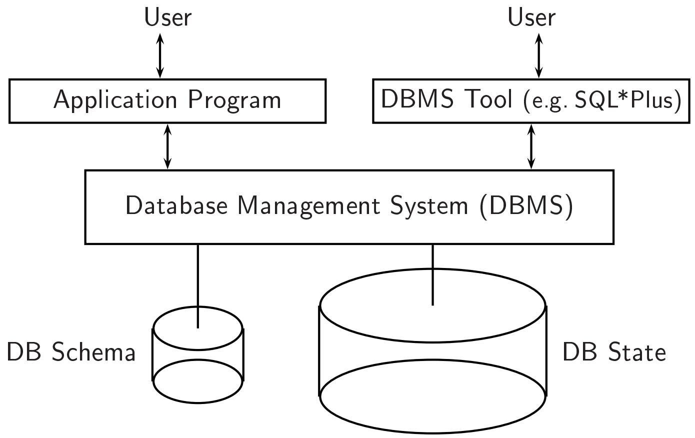
- Relational Data Model
- Entity-Relationship-Model (used in database design)
- Network Model (historical, but still in use)
- Hierarchical Model (historical, but still in use)
- Object Oriented Models
- Functional Data Model (only research systems)
- Deductive Databases (nearly only research systems)
- SGML, XML

Basic Notions (5)

Database Management System (DBMS):

- Application-independent software which implements a data model, i.e. allows
  - definition of a DB schema for some concrete application,
  - storage of an instance of this schema on a disk (or another persistent storage medium),
  - querying the current instance (database state),
  - changing the database state.
- DDL: Data Definition Language (for defining a schema).
- DML: Data Manipulation L. (for querying/updating the state).

Basic Notions (6)

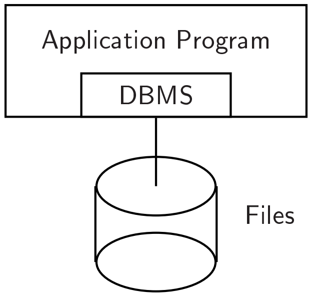


Overview

1. Persistence, Schema & State
2. Data Independence, Three-Schema Architecture
3. Other DBMS Functions, e.g. Transactions
4. Database Users and Database Tools
5. Summary

DBMS as a Subprogram Library (1)

- Most DBMS use operating system files to store the data.
- But the DBMS gives you higher level operations, which you can use in your own application programs.



DBMS as a Subprogram Library (2)

- For instance, a DBMS contains routines for
  - Sorting (Mergesort)
  - Searching (B-Trees)
  - File Space Management, Buffer Management
- Optimized for large data sets (larger than main memory).
- Reusing well-debugged code in the DBMS can save a lot of programming efforts!

Physical Data Independence (1)

- The DBMS is a layer of software above the OS files.
- You can access the files only via the DBMS.
- Indirection gives the possibility of hiding internal changes.
- Idea of abstract data types:
  Change the implementation, but keep the interface.
- Here the implementation is mainly the file structure, which has to be changed for performance reasons.
  But the application program interface is kept stable.

Physical Data Independence (2)

Typical Example:

- At the beginning:
  - Frequent queries for the account number of a given owner were not expected.
  - The accounts file was small, so that a sequential search (full table scan) was sufficiently fast.
- Later this kind of query happened quite often and the file became larger (the “system workload” has changed).
- Thus: An index (e.g. B-tree) is needed for faster access.

Physical Data Independence (3)

Without DBMS (or with a Pre-Relational DBMS):

- Using the index to access the file must be explicitly mentioned, so application programs must be changed.

With Relational DBMS:

- The system hides the existence of indexes at the interface.
- So queries and updates do not have to (and cannot) refer to the index.
- The system automatically
  - modifies the index in case of updates,
  - uses the index to evaluate queries when advantageous.

Physical Data Independence (4)

Conceptual Database Schema (“Interface”):

- Only logical information content of the database
- Simplified View: Some complicating details are hidden

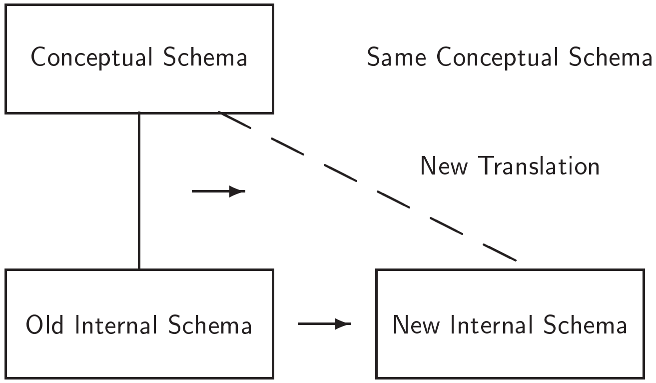
Internal/Physical Database Schema (“Implementation”):

- Indexes
- Division of tables among disks
- Storage management parameters if tables grow or shrink
- Physical placement of new rows in a table (“clustering”)

Physical Data Independence (5)

1. The user enters a query referring to the conceptual schema.
2. The system translates this user query into a query (“execution plan”) which refers to the internal schema.
3. The system executes the translated query (“execution plan”) on the stored instance of the internal schema.
4. The system translates the result back to the conceptual level.

Physical Data Independence (6)



Declarative Query Languages (1)

- Physical data independence means that the query language cannot refer to indexes.
- Declarative query languages go one step further:
  - They should describe only what information is sought,
  - but should not prescribe any particular method how to compute this information.
- Algorithm = Logic + Control (Kowalski)
  Imperative/Procedural Language: Explicit Control, Implicit Logic.
  Declarative/Descriptive Lang.: Explicit Logic, Implicit Control.

Declarative Query Languages (2)

- SQL is a declarative query language:

```
SELECT  X.No
FROM    ACCOUNT X
WHERE   X.Owner = 'Ullman, J.D.'
```

- Often simpler formulations:
  (the user does not have to think about efficient execution)
  Often much shorter than imperative programming:
  ⇒ Less expensive program development and maintainance.

Declarative Query Languages (3)

- Declarative query languages
  - allow powerful optimizers
    (because they don't prescribe a query evaluation method)
  - need powerful optimizers
    (because the naive evaluation algorithm would be too inefficient)
- Larger independence of current hardware/software technology.
  - Simpler Parallelization
  - Today's queries can later use tomorrow's algorithms!

"Data Independence"

- Decoupling between programs and data.
- Data is an independent resource by itself.
  - Earlier it had meaning only in the context of the application programs for which it was originally collected.
- Physical data independence:
  - Programs should not depend on data storage methods.
  - Vice versa, the file structures are not determined by the programs.

  ⇒ Protects investments in application programs
  (need fewer changes / are longer usable).

Logical Data Independence (1)

- Logical data independence allows changes to the logical information content of the database.
- Of course, information can only be added.
  E.g. add a field "`CreditLimit`" to the `ACCOUNT`-Table.

  Or represent information differently, e.g. use 24h-notation instead of AM/PM.
- This may be required for new applications.
- It should not be necessary to change old applications, although the records are now longer.

Logical Data Independence (2)

- Logical data independence is only important when there are application programs with distinct, but overlapping information needs.
- Logical data independence also helps to integrate previously distinct databases.
  - In earlier times, every department of a company had its own database / data files.
  - Now, businesses generally aim at one central database.
    It may be distributed, but this is another issue.

Logical Data Independence (3)

Advantages of Central Databases:

- No redundancy (duplicate storage of the same information).
- No duplicate database administration.
- Allow queries using information from multiple sources.

Disadvantages of Redundant Storage of Information:

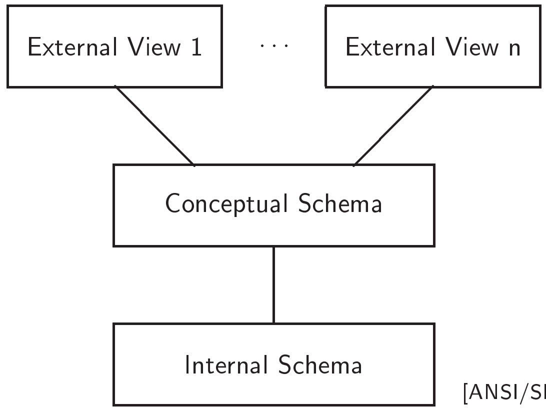
- Duplicates the efforts for keeping information up-to-date.
- Leads to inconsistencies:
  Two copies of the information do not agree.
- Wastes storage space (also on backup tapes).

Logical Data Independence (4)

External Views:

- Logical data independence requires a third level of database schemas, the external views.
- Each database user can have an individual view of the data.
- An external view contains a subset of the information in the database, maybe slightly restructured.
- In contrast, the conceptual schema describes the complete information content of the database.
- External views are also important for security reasons.

Three-Schema Architecture



[ANSI/SPARC 1978]

Overview

1. Persistence, Schema & State
2. Data Independence,
   Three-Schema Architecture
3. Other DBMS Functions, e.g. Transactions
4. Database Users and Database Tools
5. Summary

Transactions (1)

- A transaction is a sequence of database commands, especially updates, which the DBMS treats as a unit.
- E.g. a transfer of 50 dollars from account 1 to account 2 consists of
  – checking the current balance/credit limit of account 1,
  – decreasing the balance of 1 by 50 (debit),
  – increasing the balance of 2 by 50 (credit).

ACID-Properties:

- Atomicity, Consistency, Isolation, Durability.

Transactions (2)

Atomicity:

- Modern DBMS guarantee that a transaction is
  – either executed in total,
  – or not at all.
- If the transaction cannot be executed until the end (e.g. because of a power failure or system crash), the database state before the transaction has begun will be restored when the DBMS is started the next time.
- As long as the transaction has not been declared as complete (with `COMMIT`), all changes can be undone (`ROLLBACK`).

Transactions (3)

Durability:

- When a DBMS acknowledges the successful end of a transaction, it guarantees that the changes are durable.
- The changes are stored on disk — they are not lost even if there is a power failure one second later.

  In operating systems, you often cannot be sure whether the data is on disk or still in a buffer.
- A larger DBMS has powerful mechanisms for backup and recovery, such that even if a disk fails, no data is lost.

  Operating system backup utilities typically create only one backup per day. All work after the last backup will be lost.

Transactions (4)

- So atomicity and durability mean together that there is one point in time which lies between
  - the user telling the system that the transaction is complete (`COMMIT`), and
  - the system telling the user that this command was successfully processed,

  when all changes become effective.
- If the system crashes before this point, the database state is not changed.
- If the system crashes after this point, the database state contains all changes which the transaction has executed.

Transactions (5)

Isolation:

- A larger DBMS supports concurrent accesses of multiple users.
- Without control, this could have strange effects including lost updates.
- However, a DBMS tries to create the impression that every transaction runs in isolation, i.e. has exclusive access to the complete database.
- Usually, a DBMS automatically manages locks on DB objects for this purpose.

Transactions (6)

Consistency:

- User and system can be sure that the current state is the result of a sequence of completely executed transactions.
- For instance, it is not possible that a tuple was entered into a table, but not into the index on this table because of a power failure in between.
- The user must ensure that each transaction, if applied fully and isolation to a consistent state, will produce a consistent state. Modern DBMS offer some support (see below).

  So the consistency is in part consequence of the other three properties and in part user responsibility. However, it should be appreciated that all data structures are kept consistent.

Integrity Enforcement

- There can be errors when data is entered into the database, i.e. the data does not correspond to that part of the real world which is represented in the database.
- Such invalid data tends to accumulate over time and makes the database useless in the end.
- A good DBMS can automatically check certain conditions on the data.
- A good database schema should contain enough of these "integrity constraints" or "business rules" such that every instance corresponds to a possible state of the real world.

Access Control

- Not all users who have access to a database have the same rights (i.e. not every user should be able to perform queries/updates on all tables).
- A larger DBMS has powerful mechanisms for access control: Which user may perform which operations on which objects?
- A larger DBMS can create also a protocol of selected actions of selected users.
- Often, there is a database administrator (DBA) who may do everything.
- In large enterprises, even the administration of the database is divided among multiple users (with different rights).

Data Dictionary (1)

- The database schema is usually stored in the database itself (in predefined system tables).
- Then, the schema can be queried with the same powerful language as the data.
- A good DBMS makes all its data available via its query language:
  - Tables, Columns, Views
  - Indexes, Physical Storage Information
  - Database Users, Access Rights
  - Statistical Information, Performance Data

Data Dictionary (2)

SYS_TABLES

TABLE_NAME	OWNER
ACCOUNT	BRASS
SYS_TABLES	SYS
SYS_COLUMNS	SYS

SYS_COLUMNS

TABLE_NAME	SEQ	COL_NAME
ACCOUNT	1	NO
ACCOUNT	2	OWNER
ACCOUNT	3	BALANCE
SYS_TABLES	1	TABLE_NAME
SYS_TABLES	2	OWNER
SYS_COLUMNS	1	TABLE_NAME
SYS_COLUMNS	2	SEQ
SYS_COLUMNS	3	COL_NAME

A real system will store more information, e.g. the column data type.

Disadvantages of Database Systems

- Expensive
- Dependence on the DBMS supplier.

  DBMS software often runs only with a specific OS release. So if the OS is updated, the DBMS may have to be updated, too. Both are tightly coupled because they perform in part similar tasks.

  Of course, there are Standards for SQL, but every system has extensions and special tools.
- Requires quite a lot of time to learn.

  Oracle 8 has $\geq$ 95 volumes (= 1.70 m) of documentation.
- Overhead: A hand-optimized C-routine is faster than the general purpose code in the DBMS.

Arguments not to use a DBMS

- The data will be processed by a single program. No other applications on the same data are planned.
- A DBMS seems unusable for various reasons, e.g.:
  - Response time must be very short (realtime),
  - non-standard locking needed.
- You can afford the redevelopment:
  - The structure of the data is simple.
  - All data fits into main memory.

Exercise

- Take the task of developing a system where students can vote about the quality of lectures.
- There is a form in the world wide web, where students can enter their data, and when they press the “submit” button, these data are stored on the web server.
- Later evaluation is performed on the collected data, e.g. computing average values.
- I have proposed writing C programs for this task, and storing the data in UNIX files.
- What arguments can you make as to why using a database system might be better?

Overview

Database Users (1)

Database Administrator:

- Knows the complete database schema well.
- Changes the DB schema if needed, documents such changes.
- Gives access rights to database users. Ensures security.
- Monitors system performance. Does performance tuning.
- Monitors available disk space and installs new disks.
- Ensures that backup copies of the data are made. Does the recovery after disk failures etc.
- Installs new versions of the DBMS software.
- Tries to ensure data correctness.

Database Users (2)

Database Administrator, Continued:

- Responsible for licence agreement
- Contact for support / DBMS vendor
- Expert on the DBMS software
- Sometimes needs powerful privileges for the operating system
- Can damage everything.

Database Users (3)

Application Programmer:

- Writes programs for standard tasks (for "naive users").
- E.g. a program for a debit or credit, which asks the teller for an account number and an amount and then does the necessary changes in the DB.
- Develops a web interface for the database if needed.
- Knows the database language SQL well, plus some programming languages and development tools.
- Usually supervised by the database administrator.

  Or outside consultant who leaves after the project is finished.
- Might do database design (there are specialists for this task).

Database Users (4)

Sophisticated User (one kind of "End User"):

- Knows SQL and/or some query tools.
- Does non-standard evaluations of the data without help from application programmers.
- E.g. some kind middle manager who needs some new statistics for decision support.

Naive User (the other kind of "End User"):

- Communicates with the DB only via application programs.
- Does the real work of entering data.

Database Tools

- Database Server (often a set of background processes)

  Like e.g. a web server: It gets requests over the net (from a client), processes them, and sends an answer back.
- Interactive SQL interpreter
- Graphical/Menu-based query tools
- Interface for DB access from standard programming languages (C, Pascal, Java, ...)
- Tools for developing/using form-based DB applications (4GL)
- Report generator
- WWW interface
- Tools for Import/Export, Performance Monitoring, ...

Knowing Oracle (1)

Core:

- Relational Model, Fundamentals of Database Design.
- SQL: SQL-92 (current standard) plus Oracle deviations
- Data Types and Data Type Functions.
- Data Dictionary
  Lists all DB objects — all information Oracle has.
- PL/SQL: Oracle's programming language, tightly coupled with SQL (e.g. for server-side stored procedures, triggers).
- SQL*Plus: Output Formatting, Scripts
- Orientation in the Oracle Documentation: Where to look.

Knowing Oracle (2)

Database Administration:

- SQL: Oracle Extensions, e.g. for physical storage parameters, user management.
- Oracle Process Structure, Installation Parameters, Tuning, Internal Data Structures.
- Administration Tools: Starting and Stopping the DB Server, Adding Disk Space, etc.
- Backup & Recovery in Oracle
- Import/Export-Utilities, SQL*Loader
- More Database Design, Oracle Designer/2000.

Knowing Oracle (3)

Application Programming:

- Oracle Pro*C: SQL embedded in C
- Oracle Developer: Visual programming tools for creating end-user application programs.
  E.g. Form Builder: Creates programs which are something like specialized editors for specific relations, e.g. showing one record at a time in a screen mask. Oracle Developer contains also Report Builder, Graphics Builder, etc.
- Oracle Application Server, Tools for Creating Web Interfaces.
- Java-Interfaces: JDBC, JSQL ($\rightarrow$ Oracle8i)
- Some knowledge about security, e.g. for web interfaces.

Overview

1. Persistence, Schema & State
2. Data Independence, Three-Schema Architecture
3. Other DBMS Functions, e.g. Transactions
4. Database Users and Database Tools
5. Summary

Summary (1)

Functions of Database Systems:

- Persistence
- Integration / No Redundancy
- Data Independence
- Less programming effort: Many algorithms built-in, especially for external memory (Searching & Sorting).
- Ad-hoc Queries
  (I.e. when somebody gets an idea for a new query, he/she can immediately pose it. In earlier times, he/she had to ask the programming department for a new program.)

Summary (2)

Functions of Database Systems, continued:

- High data security (Backup & Recovery)
- Combinition of updates into a transaction (Atomicity)
- Multi-User (Synchronization of concurrent accesses)
- Integrity Enforcement
- Views for different users (user groups)
- Data Access Control
- System Catalog (Data Dictionary), Queries to Meta-Data
  (Meta-Data are data about data.)

Summary (3)

The main goal of the DBMS is to give the user a simplified view on the persistant storage, i.e. to make complications "transparent". The user does not have to worry about:

- Physical storage details
- Different information needs of different users
- Efficient query formulation
- Possibility of system crashes / disk failures
- Possibility of concurrent access by other users