



Why this Lecture?

Databases form the backbone of every modern information system

- A robust database management system (DBMS) is crucial for these systems.
- The knowledge of the internals of a DBMS forms the prerequisite for building and extending a DBMS as well as for building the DBMS part of a larger application in a robust fashion.
- In addition, it helps to understand the role of the different parameters of commercial DBMS and thus, tune these parameters in a way that results in a robust and performant system.

Outline

This course explores the internals of database management systems

- Address the architectures and implementation issues relevant for these systems

Complementary Topics

- Database design:
 - The informal, high-level specification of the schema of a database
 - Notations like the Entity-Relationship-Model
 - The implementation of designs in the data-definition portion of SQL
- Database programming
 - Writing Queries and database modification commands using appropriate languages, especially SQL

Literature

Hector Garcia-Molina, Jeffrey D. Ullman, Jennifer Widom:
Database System Implementation

OR

Database Systems: The Complete Book

Saake, G.: Heuer, A., K.-U. Sattler: Datenbanken:
Implementierungstechniken

Theo Härder, Erhard Rahm: Datenbanksysteme - Konzepte
und Techniken der Implementierung

Shasha, D., Bonnet, P.: Database Tuning: Principles,
Experiments and Troubleshooting Techniques

What should a DBMS achieve?

Codd's Nine Rules

- Integration: uniform, non-redundant data management
- Operations: store, search, modify
- Catalogue: access database descriptions in a data dictionary
- Views for different users
- Ensuring Integrity: Correctness of the content of the database
- Data security: Rule out unauthorized access
- Transactions: Bundle several database operations to one unit
- Synchronization: coordinate parallel transactions
- Availability of data: Recover data after system failures

Megatron 3000 Implementation Details

Relations stored in files (ASCII)

e.g., relation R is in /usr/db/R

```
Smith # 123 # CS
Jones # 522 # EE
:
:
```

How NOT to implement a DBMS

Megatron 3000: An example provided by Hector Garcia Molina

Megatron 3000 Implementation Details

Directory file (ASCII) in /usr/db/directory

```
R1 # A # INT # B # STR ...
R2 # C # STR # A # INT ...
:
:
```

Megatron 3000 Sample Sessions

Implementation of DBMS

```
% MEGATRON3000
  Welcome to MEGATRON 3000!
&
.
.
& quit
%
```

WS 25/26
Frankfurt UAS

Prof. Dr. Justus Klingemann

Megatron 3000 Sample Sessions

Implementation of DBMS

```
& select *
from R #

Relation R
A      B      C
SMITH  123   CS
&
```

WS 25/26
Frankfurt UAS

Prof. Dr. Justus Klingemann

Megatron 3000 Sample Sessions

Implementation of DBMS

```
& select A,B
  from R,S
 where R.A = S.A and S.C > 100 #

A      B
123   CAR
522   CAT
&
```

WS 25/26
Frankfurt UAS

Prof. Dr. Justus Klingemann

Megatron 3000

To execute “`select * from R where condition`”:

Implementation of DBMS

WS 25/26
Frankfurt UAS

Prof. Dr. Justus Klingemann

Megatron 3000

To execute “**select * from R where condition**”:

- (1) Read dictionary to get R attributes
- (2) Read R file, for each line:
 - (a) Check condition
 - (b) If OK, display

Megatron 3000

To execute “**select A,B from R,S where condition**”:

- (1) Read dictionary to get R,S attributes
- (2) Read R file, for each line:
 - (a) Read S file, for each line:
 - (i) Create join tuple
 - (ii) Check condition
 - (iii) Display if OK

Megatron 3000

To execute “**select A,B from R,S where condition**”:

What's wrong with the Megatron 3000 DBMS? (1)

Tuple layout on disk

- e.g.,
- Change string from 'Cat' to 'Cats' and we have to rewrite file
 - Deletions are also expensive

Search expensive; no indexes

- e.g.,
- Cannot find tuple with given key quickly
 - Always have to read full relation

Brute force query processing

- e.g.,
- ```
select *
 from R,S
 where R.A = S.A and S.B > 1000
 - Do select first?
 - More efficient join?
```

No buffer manager

- e.g.,
- Need caching

## What's wrong with the Megatron 3000 DBMS? (2)

No concurrency control

No reliability

- e.g.,
- Can lose data
  - Can leave operations half done

No security

- e.g.,
- File system insecure
  - File system security is coarse

No application program interface (API)

- e.g., How can a payroll program get at the data?

No GUI

Cannot interact with other DBMSs.

Poor dictionary facilities