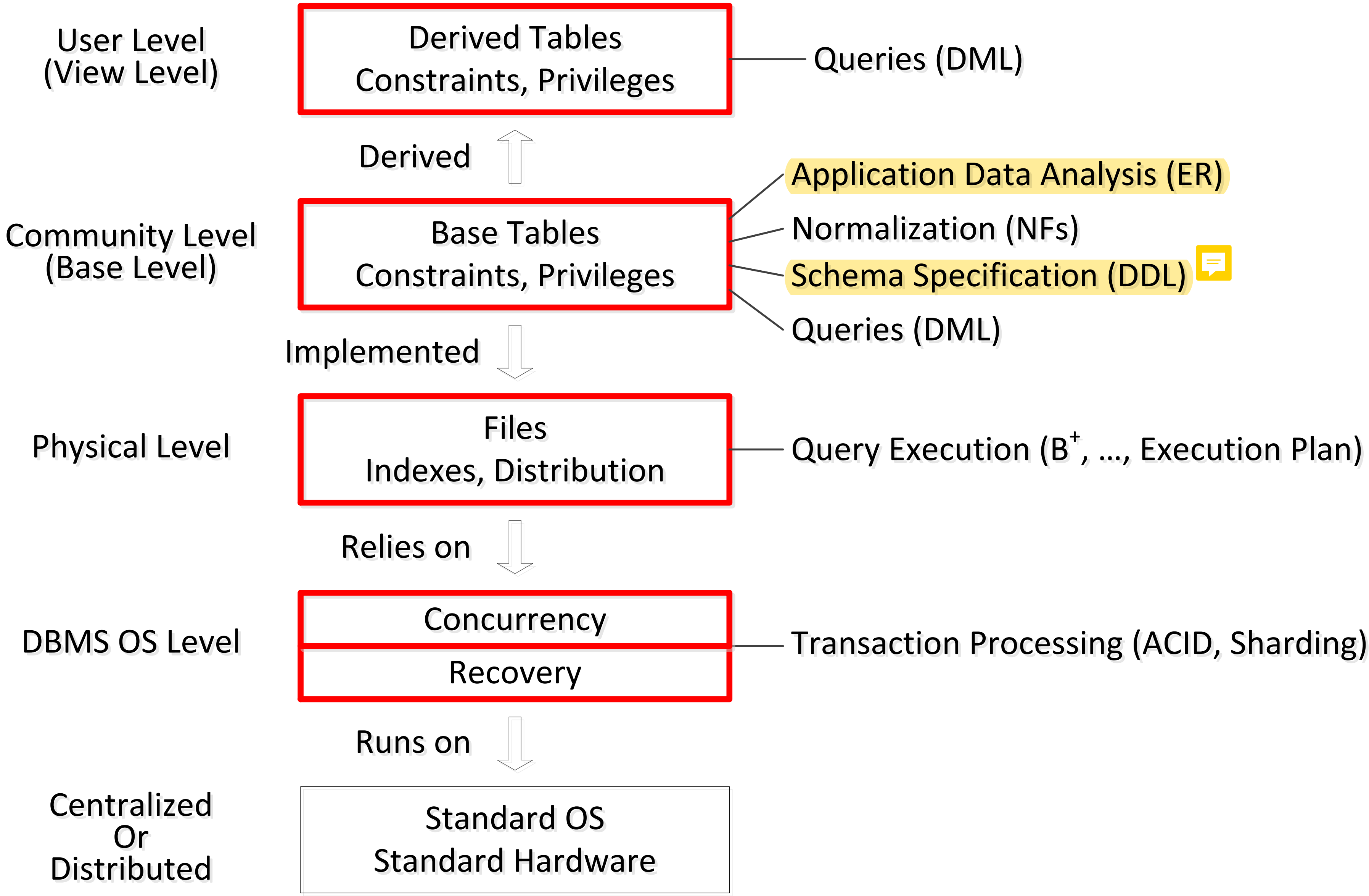
Unit 1
Goals And an Informal Synopsis of the Course

Prof Dennis Shasha
Notes from Prof Zvi Kedem
(modified by Dennis Shasha)

Introduction

Main Material Covered
We Will Understand Soon What This is



Goals of the Course

- To learn fundamental concepts of state-of-the art databases (more precisely called **database management systems**: DBMSs)
- To get to know an open-source tool for database management.
  - MySQL
- To know enough so that it is possible to read/skim a database system manual and
  - Design and implement small databases
  - Manage, query, and update existing databases
- But manuals do not teach you about some important concepts that you must know to be effective
- Understand the ideas behind recent database-like systems, such as MapReduce, Hadoop, MongoDB
- Build your own in-memory time series database system.

Two Main Functions of Databases

- A very large fraction of computer use is devoted to business processing of data using databases
  - Think about what Amazon has to do to manage its operations
- Two main uses of databases
  - ***OLTP*** (Online Transaction Processing):

    The database is used is for entering, modifying, and querying data

    Correctness, at least for entering and modifying data must be assured

    Example: Amazon charges the customer's credit card for the price of the book that the customer ordered
  - ***OLAP*** (Online Analytical Processing)

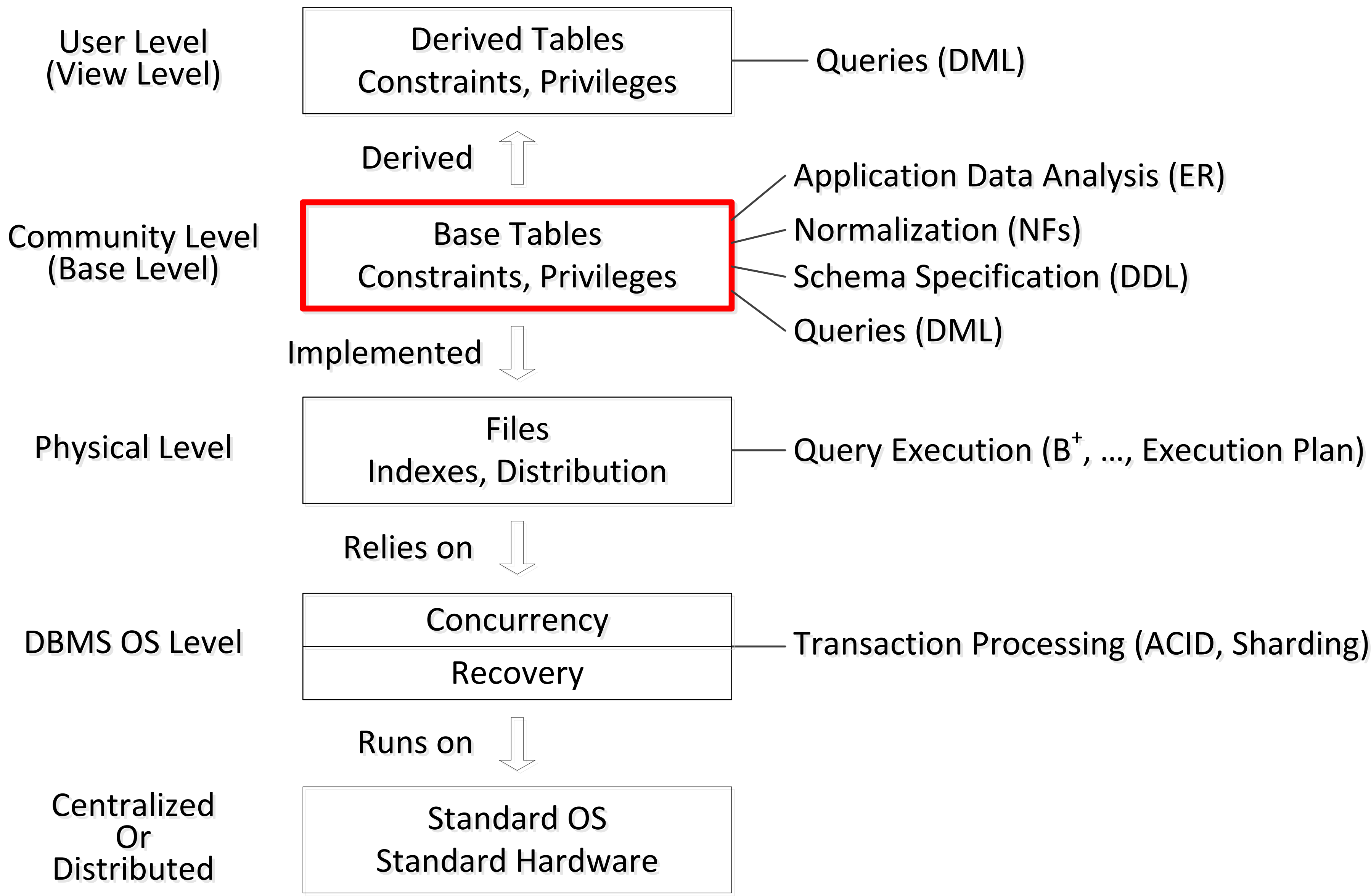
    The database is used for business intelligence, including data mining and machine learning

    Example: Amazon wants to know how many books that cost less than $10 each were sold in New Jersey during July 2019

    Example: Find out which jeans are being sold well in southern California in April.

Informal Synopsis

Topics


User Level
(View Level)
Derived Tables
Constraints, Privileges
Queries (DML)
Derived
Community Level
(Base Level)
Base Tables
Constraints, Privileges
Application Data Analysis (ER)
Normalization (NFs)
Schema Specification (DDL)
Queries (DML)
Implemented
Physical Level
Files
Indexes, Distribution
Query Execution (B+, …, Execution Plan)
Relies on
DBMS OS Level
Concurrency
Recovery
Transaction Processing (ACID, Sharding)
Runs on
Centralized
Or
Distributed
Standard OS
Standard Hardware

Managing the Data of an Enterprise

- We may consider some enterprise (organization) and the totality of the information it maintains.
- We think about managing this information, focusing on OLTP
- Ideally, the information should be stored in a (logically) single (possibly physically distributed) database system
- We start with a very simple example to introduce some concepts and issues to address
- We look at only a very small part of information of the type that an enterprise may need to keep
- We need some way of describing sample data
- We will think, in this unit, of the database as ***a set of tables***, each stored as a ***file on a disk***
- If database supports a language like "SQL", it is a relational database

A Sample Relational Database

Table1

SSN	City	DOB
101	Boston	3498
106	London	2987
121	Portland	2367
132	Miami	3678

Table2

Name	SSN	DOB	Grade	Salary
A	121	2367	2	80
A	132	3678	3	70
B	101	3498	4	70
C	106	2987	2	80

Table3

SSN	Title	Date
132	Python	8976
121	Physics	9003

Table4

SSN	Illness	Date
101	Cold	3498
121	Flu	2987

- The database stores information about employees, titles of books they checked out from the library (and when), and various illnesses they had (and when)

Some Typical Queries

- Some typical queries
  - Give Name of every employee born before 3500
  - Give Name and City for every employee who checked out any Title after 9000
  - Send a notice to every employee who had a flu to come for a checkup
- ***Note that some queries involve a single table, and some involve several tables***
- We would like to have a convenient language, as close as possible to a natural language, to express these queries, and similar ones, ***thinking of tables***, not of lower-level structures (files)
- Some languages
  - ***SQL*** (used to be called Structured Query Language): every relational database supports some version of SQL "close to standard"
  - ***QBE*** (Query By Example); underlying, e.g., Microsoft Access's GUI and some other GUIs

Two Queries in SQL

- The tables have names as shown in a previous slide (as they of course do in SQL)
  - Table1: with columns SSN, City, DOB
  - Table2: with columns Name, SSN, DOB, Grade, Salary
  - Table3: with columns SSN, Title, Date
  - Table4: with columns SSN, Illness, date
- Give Name of every employee born before 3500

```
SELECT Name
FROM Table2
WHERE DOB < 3500;
```

- Give Grade and City for every employee with the name A

```
SELECT Grade, City
FROM Table2, Table1
WHERE Table2.SSN = Table1.SSN AND Table2.Name = ′A′;
```

The Design of The Database Should Reflect the Structure of the Data

- It is important also to think carefully about the correct (or just good!) choice of which tables to use and what should be their structure
- This we should do in order to have good logical design, not worrying (yet) about efficient storage in files
- Our initial design suffers (for pedagogical reasons) from various problems, which we will see next

Redundancy

- A data item appears more than once ***unnecessarily***
  - Assuming that each SSN has only one DOB, the value of DOB appears twice unnecessarily (in two different tables)

    There is a danger that this will be inconsistent
  - Even more dangerous would have been multiple storage of employee's City

    If the employee moves, the City must be changed everywhere it appears
- Note, however, that from an efficiency point of view, it might be useful to replicate information, to speed up access
  - In our example, if frequently we want to correlate DOB with Grade and also DOB with City, it may be good to have it in both tables, and not insist on a "clean" design
- Note that ***it was necessary*** for SSN to appear in two different tables, as otherwise we could not "assemble" information about employees

Storage of Constraints (Business Rules)

- Assume that it is the policy of our enterprise that the value of Salary is determined uniquely by the value of Grade; this is a so-called ***business rule*** (semantic constraint)
  - Thus the fact that the Grade = 2 implies Salary = 80 is written twice in the database
  - This is another type of redundancy, which is less obvious at first
- There are additional problems with this design.
  - We are unable to store the salary structure for a Grade that does not currently exist for any employee.
  - For example, we cannot store that Grade = 1 implies Salary = 90
  - For example, if employee with SSN = 132 leaves, we forget which Salary should be paid to employee with Grade = 3
  - We could perhaps invent a fake employee with such a Grade and such a Salary, but this brings up additional problems, e.g.,

    What is the SSN of such a fake employee? How many employees do we have.
- The business rule specifies a pay scale, which is independent of a particular employee

Handling Storage of Constraints (Motivating Example for Normalization)

- The problem can be solved by replacing

Name	SSN	DOB	Grade	Salary
A	121	2367	2	80
A	132	3678	3	70
B	101	3498	4	70
C	106	2987	2	80

by two tables

Name	SSN	DOB	Grade
A	121	2367	2
A	132	3678	3
B	101	3498	4
C	106	2987	2

Grade	Salary
2	80
3	70
4	70

Handling Storage of Constraints (Example of Normalization)

- And now we can store information more naturally
  - We can specify that Grade 3 implies Salary 70, even after the only employee with this Grade, i.e., employee with SSN 132 left the enterprise
  - We can specify that Grade 1 (a new Grade just established) implies Salary 90, even before any employee with this grade is hired

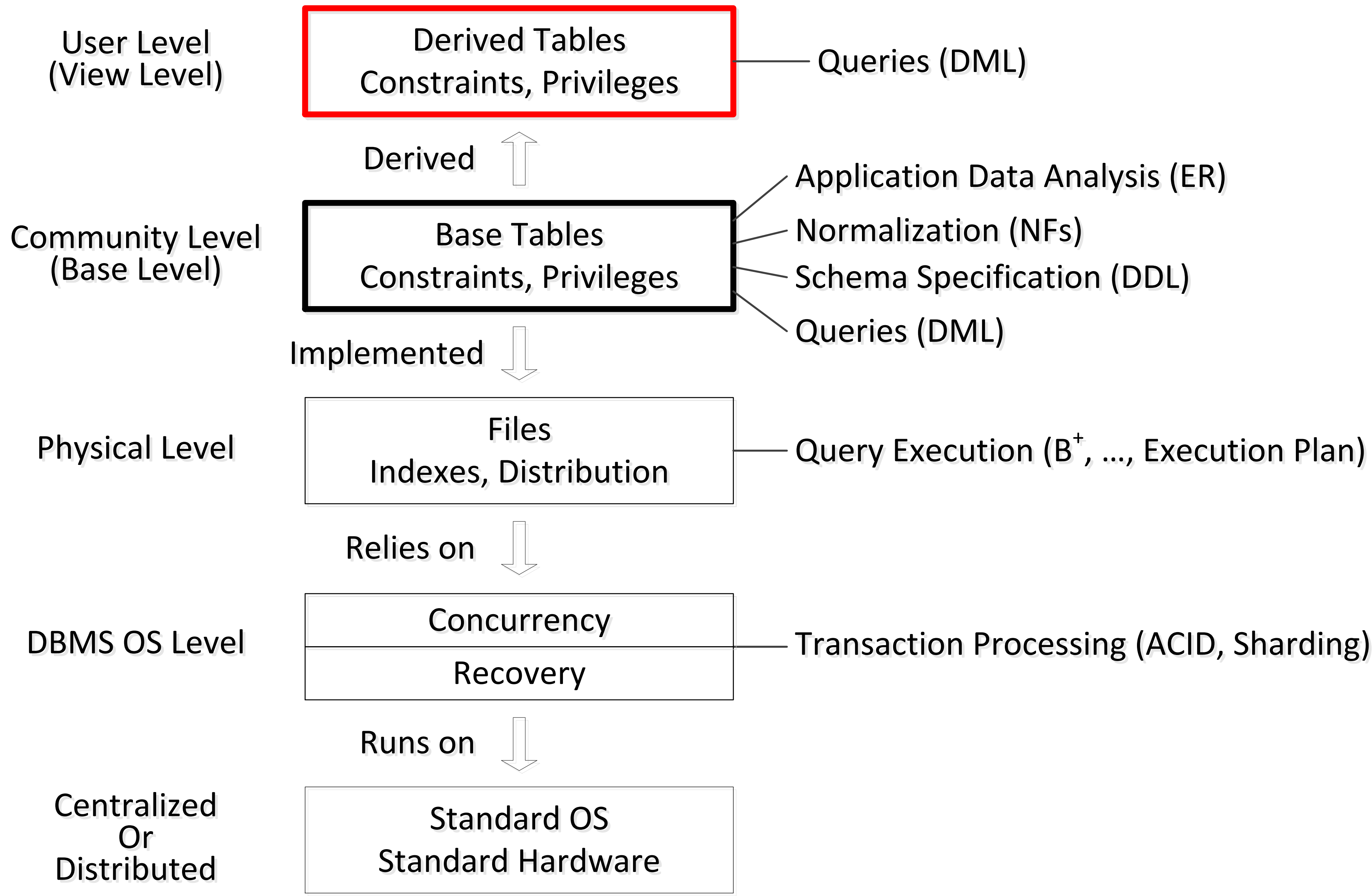
Name	SSN	DOB	Grade
A	121	2367	2
B	101	3498	4
C	106	2987	2

Grade	Salary
1	90
2	80
3	70
4	70

One More Problem

- What if it becomes illegal to use social security numbers for anything other than payroll related matters?
- We will have an incredible mess and enormous amount of work to restructure the database, unless we have designed the application appropriately to begin with
- Of course we did not know that it would become illegal to use social security numbers and it was convenient to do so, so that's what we used
- So how to be able to anticipate potential problems?
- NYU had to spend considerable effort to switch from social security numbers to University ID's
- We will discuss how to "anticipate" such problems, so such switching is painless

Topics


User Level
(View Level)
Derived Tables
Constraints, Privileges
Queries (DML)
Derived
Application Data Analysis (ER)
Normalization (NFs)
Schema Specification (DDL)
Queries (DML)
Community Level
(Base Level)
Base Tables
Constraints, Privileges
Implemented
Physical Level
Files
Indexes, Distribution
Query Execution (B+, …, Execution Plan)
Relies on
DBMS OS Level
Concurrency
Recovery
Transaction Processing (ACID, Sharding)
Runs on
Centralized
Or
Distributed
Standard OS
Standard Hardware

Different Users Need Different Data

- It may be our goal to create a design that best reflects the inherent properties of the data.
  - But, various user groups may need to look at the data assuming different structure (organization) of the data
- For privacy/security reasons we may want to give different users different access privileges to the database
  - The payroll department can see salaries but cannot see diseases.
  - The health department can see diseases but cannot see salaries.
- Users may prefer to look at different aspects of the information.
  - The payroll department may prefer to see the salary in a different currency
  - The health department may prefer to see Age instead of, or in addition to, DOB

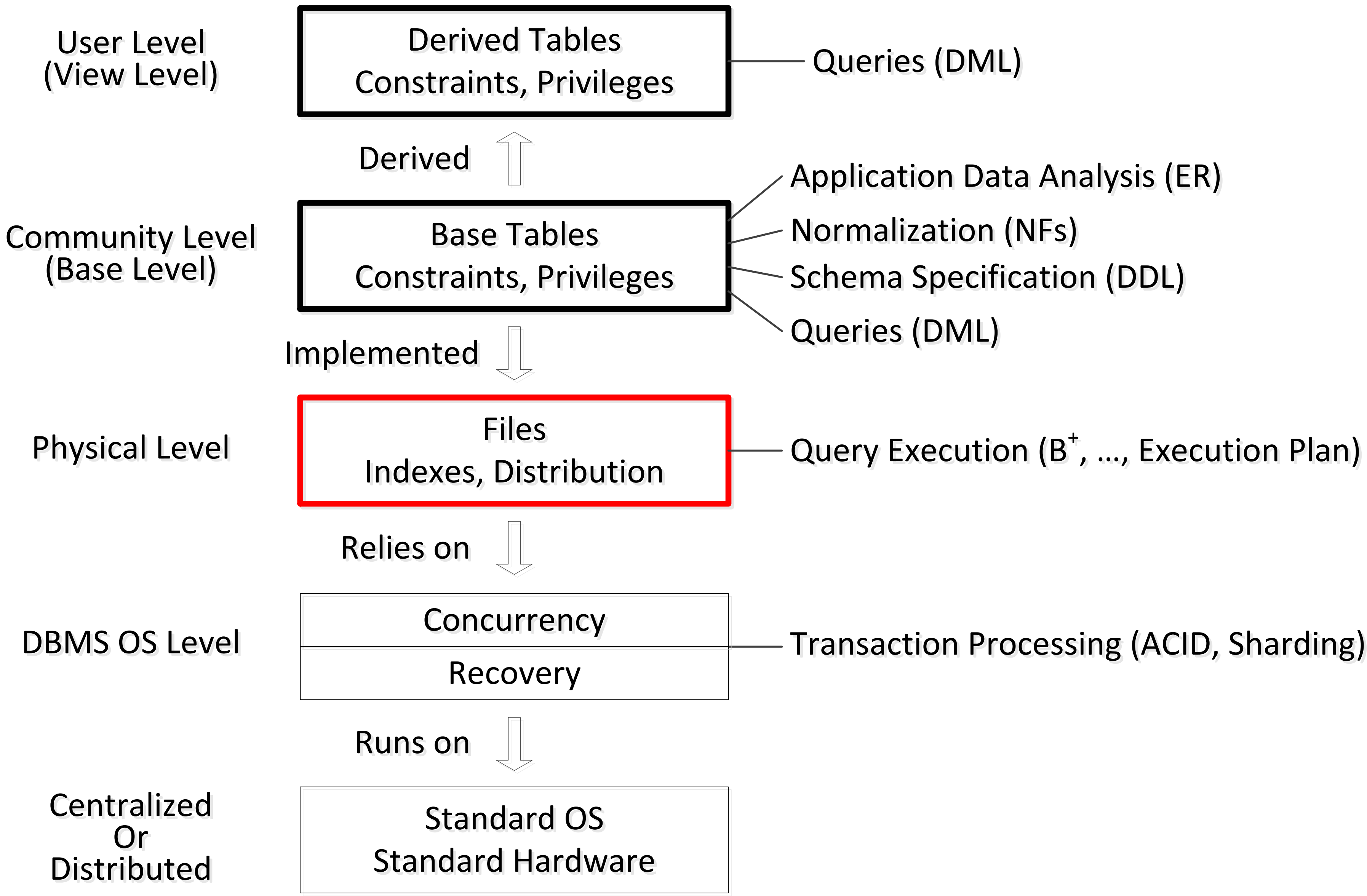
Views (Motivating Example)

- A possible solution: give each user (class of users) privileges to look at a ***view***, that is, informally speaking, a small derived database
- The health department may think that there is a table:

Name	SSN	City	DOB	Age	Illness	Date
A	121	Portland	2367	47	Flu	2987
B	101	Boston	3498	25	Cold	3721

- The database should provide such a view, which is ***computed as needed from the existing tables*** (and the current date), without the user knowing other (prohibited for this user) information
- We need to leave flexibility for unanticipated queries.
  - Some people may later be given the right and want to ask the query: “How are salaries and medical conditions correlated?”

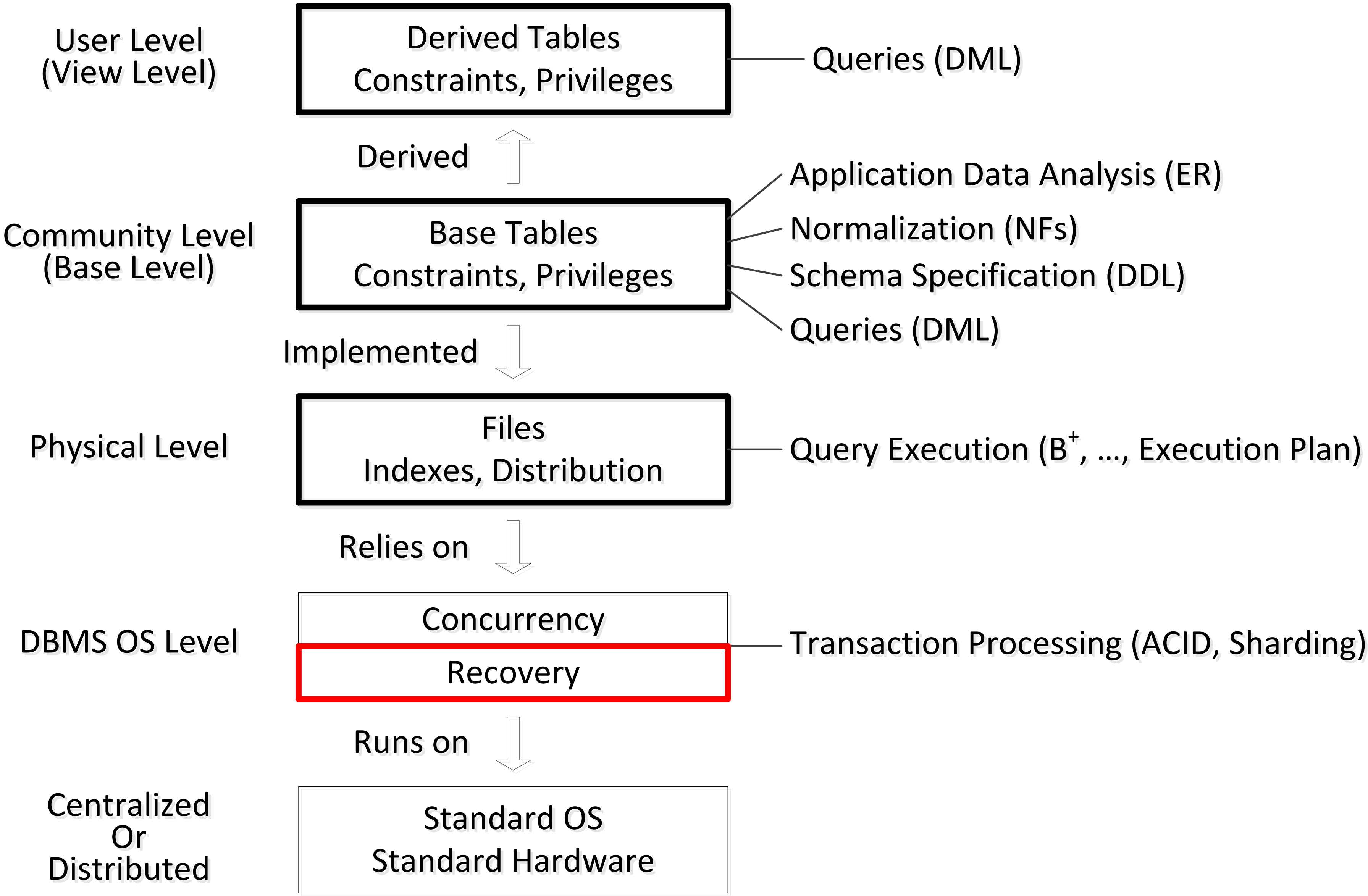
Topics


User Level
(View Level)
Derived Tables
Constraints, Privileges
Queries (DML)
Derived
Application Data Analysis (ER)
Community Level
(Base Level)
Base Tables
Constraints, Privileges
Normalization (NFs)
Schema Specification (DDL)
Queries (DML)
Implemented
Physical Level
Files
Indexes, Distribution
Query Execution (B^+, …, Execution Plan)
Relies on
DBMS OS Level
Concurrency
Recovery
Transaction Processing (ACID, Sharding)
Runs on
Centralized
Or
Distributed
Standard OS
Standard Hardware

Physical Design (With Motivating Examples)

- The database system must be organized so that it is able to process queries efficiently
- To do this:
  - ***Files must be organized appropriately***
  - ***Indexes may be employed***
- For example, if we frequently want to find the grade for various SSN, perhaps the file should be hashed on SSN, allowing direct access
- But, if we want to print the salaries of all the employees born between 2783 and 2906, maybe the file should be sorted by DOB; actually an index generalizing a 2-3 tree would generally be better
- Physical design of databases deals with such issues (including how to distribute information among various sites), which are also closely related to the optimization of query processing

Topics


User Level
(View Level)
Derived Tables
Constraints, Privileges
Queries (DML)
Derived
Community Level
(Base Level)
Base Tables
Constraints, Privileges
Application Data Analysis (ER)
Normalization (NFs)
Schema Specification (DDL)
Queries (DML)
Implemented
Physical Level
Files
Indexes, Distribution
Query Execution (B+, …, Execution Plan)
Relies on
DBMS OS Level
Concurrency
Recovery
Transaction Processing (ACID, Sharding)
Runs on
Centralized
Or
Distributed
Standard OS
Standard Hardware

Recovery (Motivating Example)

- The database must be resilient (reliable) even though the system is prone to faults.
- Assume one more table, describing employees' accounts in a credit union

SSN	Savings	Checking
101	40	30
106	40	20
121	0	80
132	10	0

- We want to give each employee a bonus of 10 in the savings account.
- To do that, a **transaction** (execution of a user’s program) will sequentially change the values of Savings

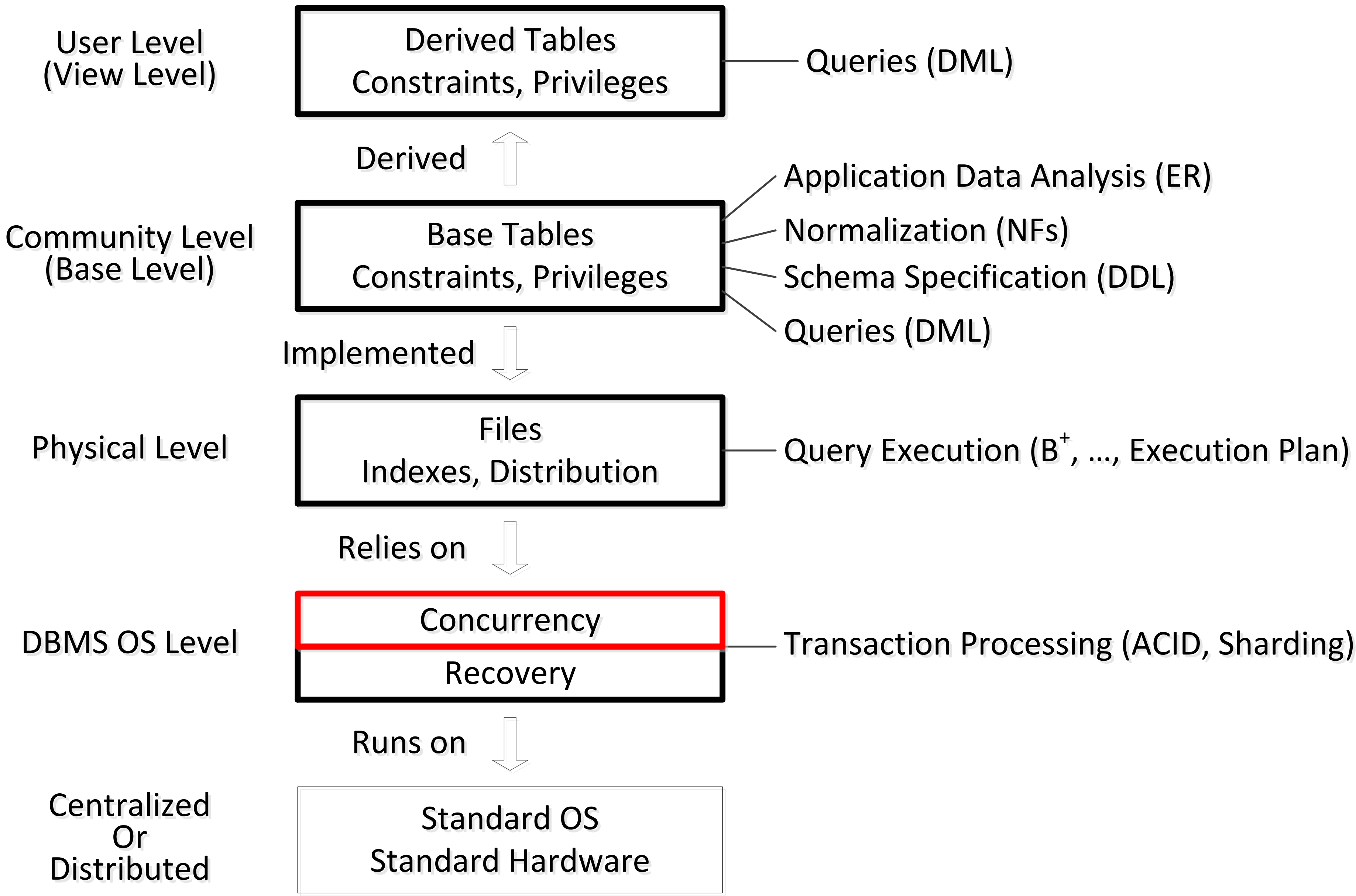
Example of A Problem

- The file describing the table is stored on a disk, values are read into RAM, modified and written out
- If X is a local variable in RAM then we have a trace of the desired execution (in shorthand):

.	
X := Savings[101]	read from disk
X := X + 10	process in RAM
Savings[101] := X	write to disk
.	

- What if the system crashes in the middle, say power goes out? Which records still need to be changed?
- Various techniques exist for managing the execution, so that reliable execution is possible

Topics


User Level
(View Level)
Derived Tables
Constraints, Privileges
Queries (DML)
Derived
Application Data Analysis (ER)
Community Level
(Base Level)
Base Tables
Constraints, Privileges
Normalization (NFs)
Schema Specification (DDL)
Queries (DML)
Implemented
Physical Level
Files
Indexes, Distribution
Query Execution (B+, …, Execution Plan)
Relies on
DBMS OS Level
Concurrency
Recovery
Transaction Processing (ACID, Sharding)
Runs on
Centralized
Or
Distributed
Standard OS
Standard Hardware

Concurrency
(Motivating Example: Updating)

- There may also be problems because of the concurrent execution of several transactions in a system
- Assume one more table, describing employees' accounts in the credit union

SSN	Savings	Checking
101	40	30
106	40	20
121	0	80
132	10	0

- Using transaction T1, we want to multiply the value of each account by 2
- Concurrently, using transaction T2, employee SSN = 121 wants to move 10 from his Checking account (CH[121] for short) to his Savings account (SA[121] for short)

Execution Trace of Two Transactions On Accounts of SSN = 121

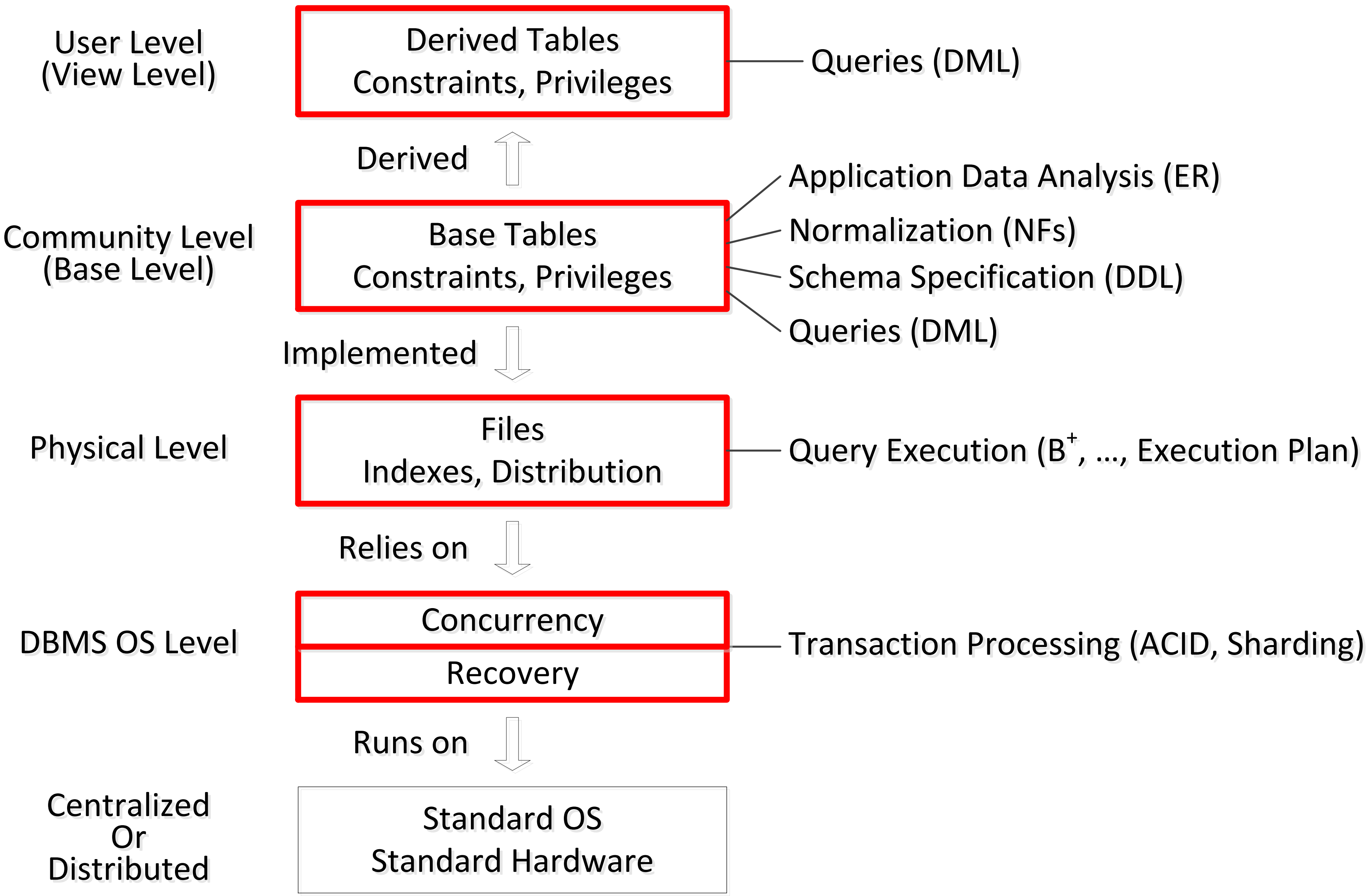
T1	T2
……..	
	Y1 := CH[121] (Y1 = 80)
	Y1 := Y1 − 10 (Y1 = 70)
	CH[121] := Y1 (CH[121] = 70)
X1 := CH[121] (X1 = 70)	
X1 := X1 * 2 (X1 = 140)	
CH[121] := X1 (CH[121] = 140)	
X2 := SA[121] (X2 = 0)	
X2 := X2 * 2 (X2 = 0)	
SA[121] := X2 (S[121] = 0)	
	Y2 := SA[121] (Y2 = 0)
	Y2 := Y2 + 40 (Y2 = 10)
	SA[121] := Y2 (SA[121] = 10)
……..	

- CH[121] + SA[121] = 140 + 10 = 150 and the SSN = 121 has the total of 150 and not 160 as he should have had

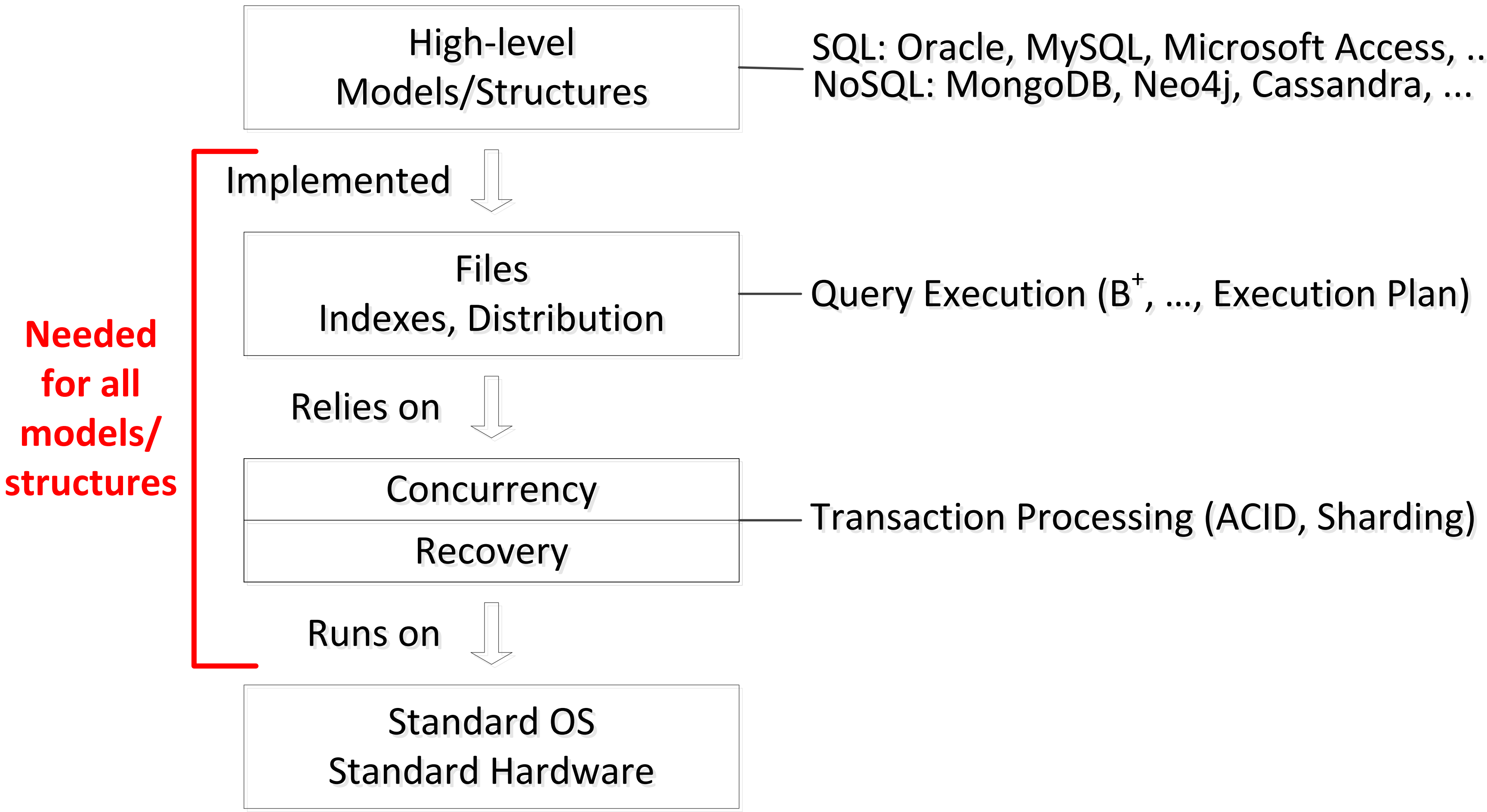
Corruption of Database Must Not Be Allowed

- The execution corrupts the database and must not be allowed
- Therefore concurrent execution of transactions must be managed so that the resulting database is correct
- Various methods to do that exists and modules implementing them are part of the system
- But, of course, the topic is not applicable to single-user system, such as Microsoft Access

The Layers


User Level
(View Level)
Derived Tables
Constraints, Privileges
Queries (DML)
Derived
Application Data Analysis (ER)
Normalization (NFs)
Community Level
(Base Level)
Base Tables
Constraints, Privileges
Schema Specification (DDL)
Queries (DML)
Implemented
Physical Level
Files
Indexes, Distribution
Query Execution (B+, …, Execution Plan)
Relies on
DBMS OS Level
Concurrency
Recovery
Transaction Processing (ACID, Sharding)
Runs on
Centralized
Or
Distributed
Standard OS
Standard Hardware

Lower Levels Needed All DBMSs Not Only For Relational DBMSs



The Layers/Levels of the Ideal Database

- It is customary to think of the database as made of several layers or levels, which are not completely standardized
- Different levels have different roles
- We will think of 4 levels:

• **User** (View, External)	Various user views
• **Community** (Base)	Description of the enterprise
• **Physical** (Internal)	Files, access methods, indexes, distribution
• **Database O.S.**	Recovery and concurrency

- The database, in general, does not run on a bare machine
- The Database O.S. (DBOS) runs on top of the O.S., such as Windows or Linux

The Community Level

- The community level is most fundamental as it describes the total information and its structure/meaning
  - to the extent we understand the information and know how to express our understanding
- It is also generally used for manipulating the database, that is querying and modifying it
- The main tools we have:
  - ***Data Definition Language*** (DDL), for specification
  - ***Data Manipulation Language*** (DML), for querying and modifying
- Tables in our example (their structure, not the specific values which change in time) were a kind of DDL
  - They form a ***schema***, a description of the structure.
- Of course, this level changes as the needs of the enterprise change

The User Level

- The user level is seen by various users
- Each view (subschema) is like a small conceptual level.
- It can also change in time.
- A particular view may be modified, deleted, or added even if the community level does not change
  - For example, it may become illegal for some user to see some information

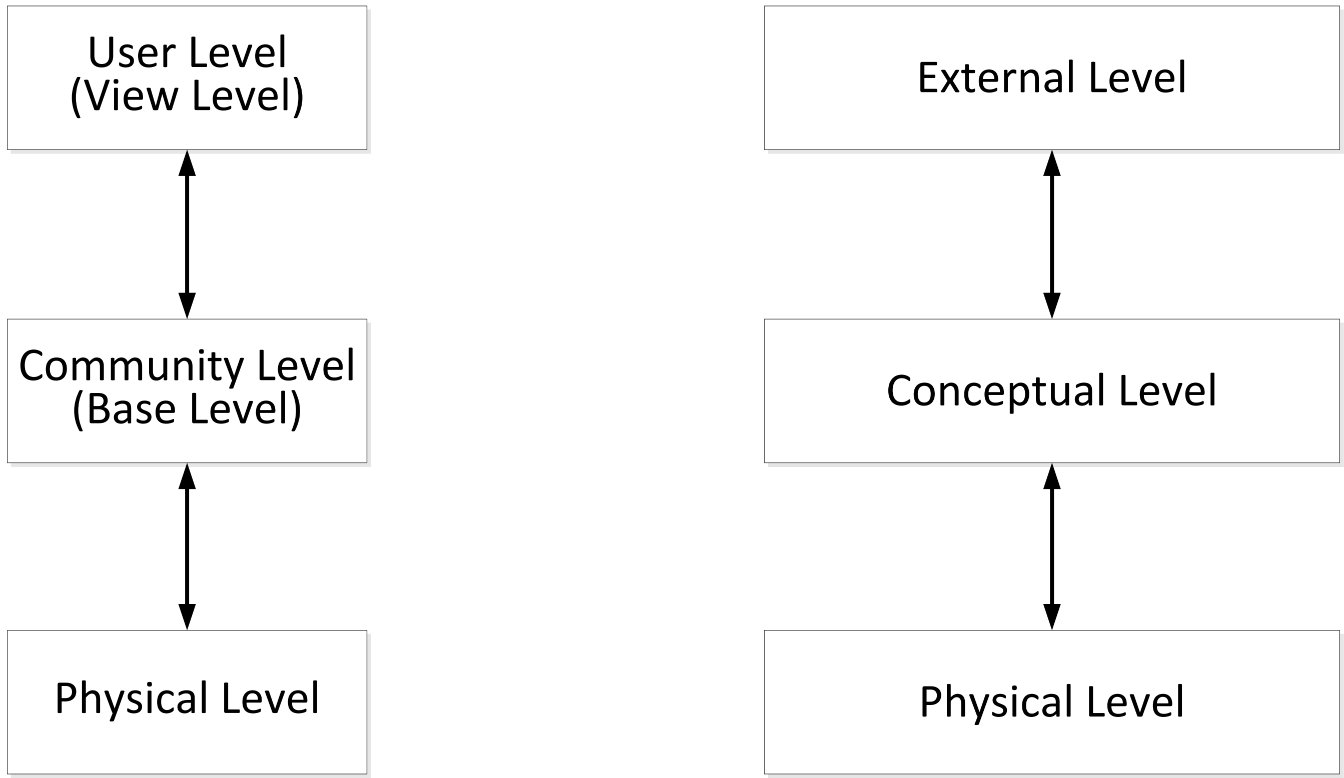
The Physical Level

- The internal level deals with file organization/storage management
- It changes in time too
  - New storage devices are brought
  - Files may have indexes created because some queries have become more frequent
  - The data may be geographically distributed

The Data Base Operating System Level

- The data base operating system level deals with concurrency and recovery
- The data base operating system can change too
- The vendor of the data base may invent better methods to handle recovery/concurrency

Another Way of Naming: Three-Tier Architecture



- Terminologies for the top 3 layers/levels/tiers
- Note that DBOS level is not normally included because once a database management system is deployed in an enterprise, the DBOS level is “not manipulated”
  - But we will still learn the fundamentals of how it works so that you know how to specify levels of correctness, and similar

Independence Among Levels

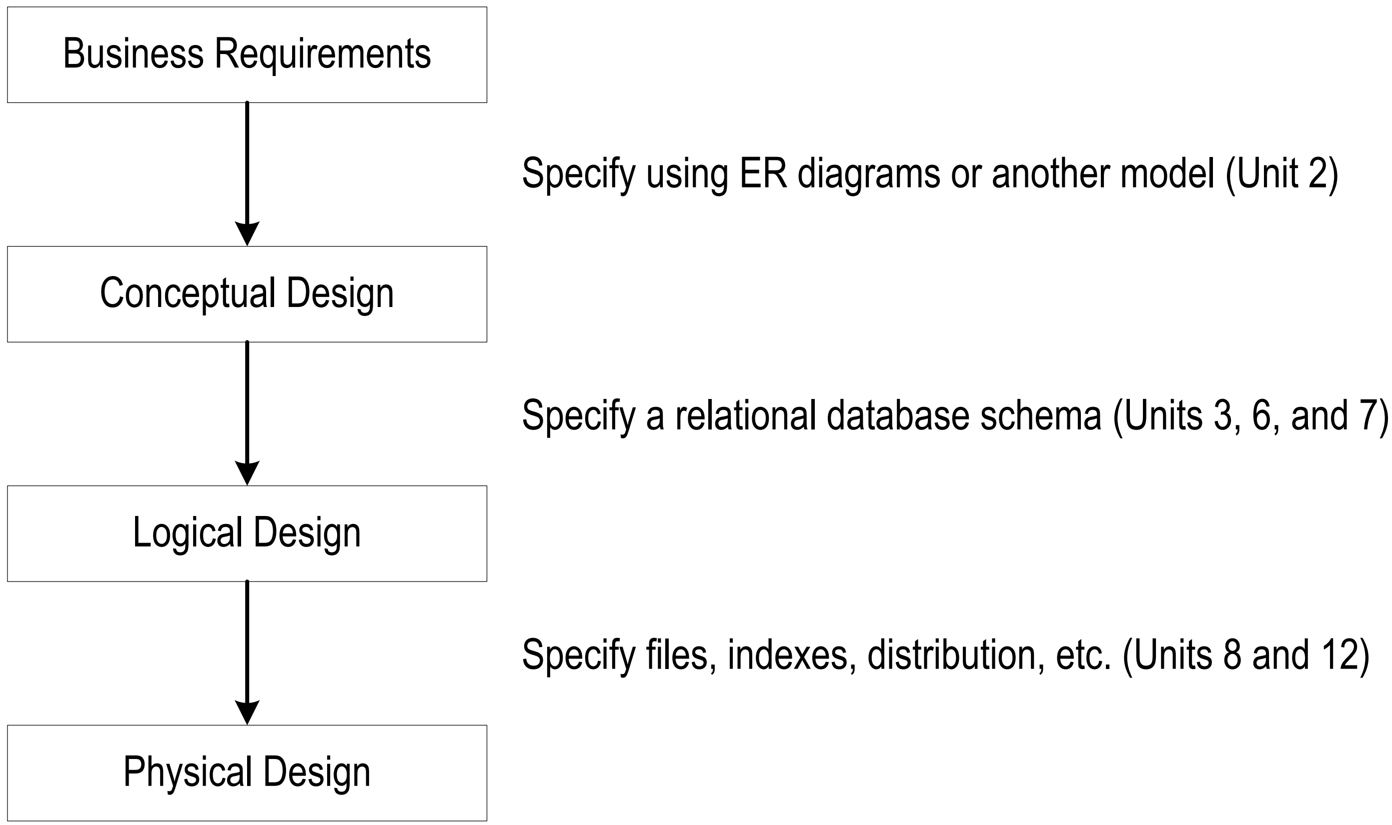
- A very important goal is (Data-) independence between/among levels
- We must make sure that changes in one level disturb as little as possible the other levels (propagate as little as possible)

Who Does What?

- The ***database vendor/implementer*** sends:
  - The database operating system
  - Tools to create and manipulate the three top levels: view, community, and physical
- The ***database designers*** discuss with the users what information the database should contain and its structure
  - A common model (language for describing reality) is needed for them to communicate

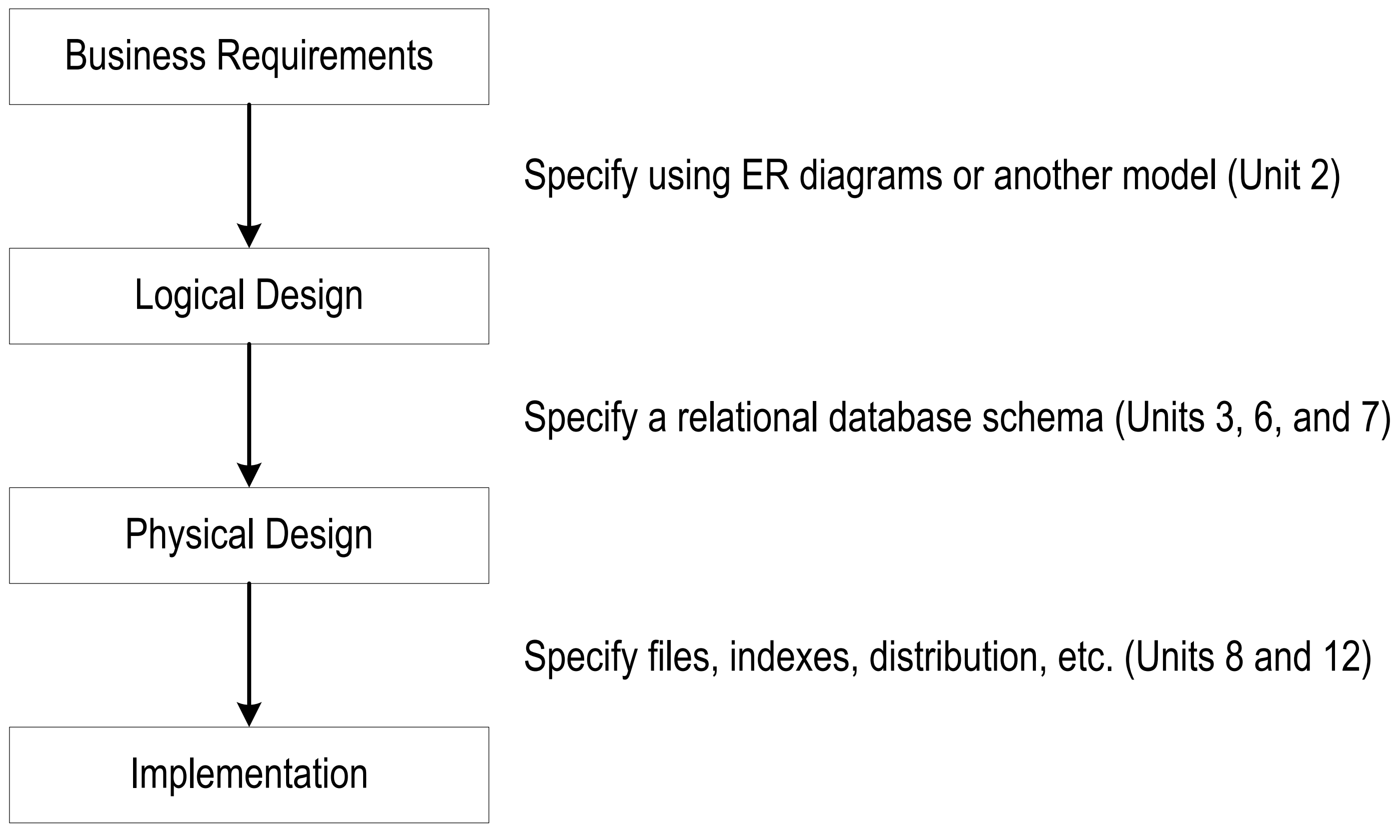
    Entity-relationship model is frequently used
- The ***database application developers*** write the programs (in ***SQL*** and other languages) that specify and manipulate the database
  - Some physical design
- The ***database administrator (DBA)*** maintains the database itself (not the specific application programs), including
  - Loading of data, some physical design, backups, tuning, etc.

Simplified Workflow
Our Terminology



- Of course, we will cover other topics too
  - SQL as Data Manipulation Language
  - DBOS
  - . . .

Simplified Workflow
Another Terminology



Challenges

- We have seen just the tip of the iceberg of what needs to happen for database systems to function as required
- We need
  1. Natural semantics
  2. Convenient syntax
  3. Efficiency
  4. 100% correctness and high reliability
- Enormous effort has been spent since the early 1970s to achieve that
- Most of the effort was on 3 and 4, but it’s not visible as it was “under the hood”

NoSQL: Some Recent Developments

- **Not SQL** or **Not Only SQL**
- Need to handle data that is not easily modeled/stored using tables
- Need much higher processing throughput
  - In tens of microseconds (or even less) for stock trading
- Distribute the computations and data accesses among multiple “nodes” to speed up processing and increase reliability

- However building large, fast, correct, and reliable distributed databases is not practical for the vast majority of applications
- So give up “some” correctness and reliability to gain speed

Material Covered

- Methodology used for modeling a business application during database design process, focusing on entity-relationship model and entity relationship diagrams
- Relational model and implementing an entity relationship diagram in it
- Relational algebra (using SQL syntax)
- SQL as data manipulation language
- SQL as data definition and data control language
- Refining a relational implementation, including the normalization process and the algorithms to achieve normalization

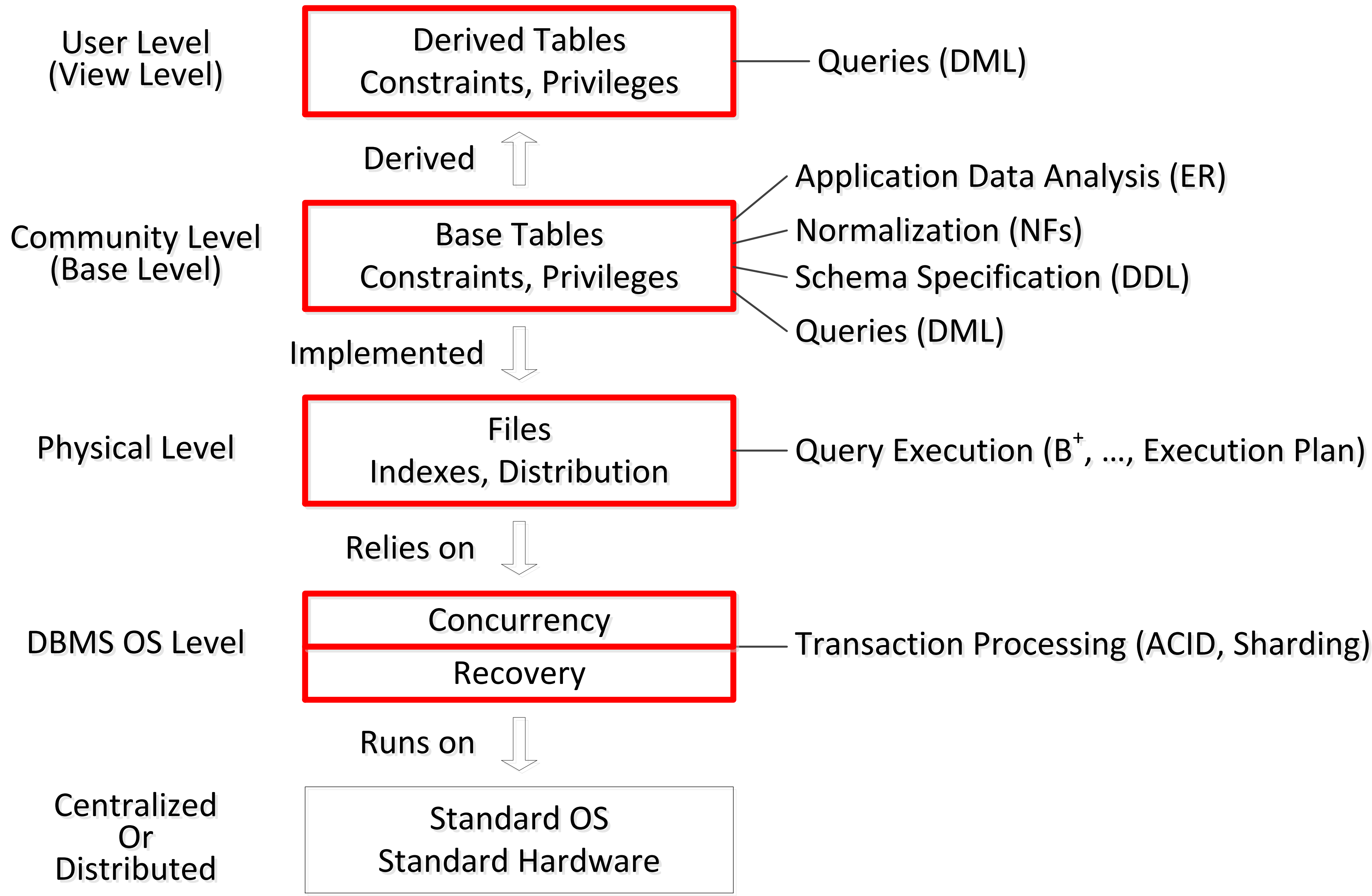
Material Covered

- Physical design of the database using various file organization and indexing techniques for efficient query processing (fundamentals and as implemented)
- Recovery (fundamentals and as implemented)
- Concurrency Control (fundamentals and as implemented)
- Query Execution
- Data warehouses
- Online analytical processing (OLAP)
- New systems

- Power Point presentations will include more material than will be covered in the course
- It will be clearly indicated what has been covered

Recapitulation

We Will Cover the Material In Red Boxes Particularly Stressing the Community Level



An Important Note: Wider Applicability Of Some Of The Material

- The material covered at the levels below the Community Level is applicable to Database Management Systems beyond Relational Database Management Systems, such as NoSQL systems

Key Ideas

Key Ideas

- ***Synopsis*** of the course
- The need for database management systems
- Brief overview of the relational model
- Querying relational database directly and through views
- Need for good logical design
- Need for good physical design
- Recovery
- Concurrency
- Layers of database management systems
- Independence between/among layers
- Various roles of designers, developers, administrators
- NoSQL systems