Ch 1. Introduction to DBMS

Sang-Won Lee

http://icc.skku.ac.kr/~swlee

SKKU VLDB Lab.

(http://vldb.skku.ac.kr/)

What is a DBMS?

- Database: A very large, integrated collection of data.

- Database models real-world *enterprise.*
  - Entities (e.g., students, courses, instructors, classroom, time)
  - Relationships (e.g., Gil-dong takes DB-41, and Lee teaches DB-41. The classroom of DB-41 is 21514 and its class hour is Mon. 1-3PM ...)
    - ✓ Recommended reading: “The power of relationships in data”
      - https://www.allthingsdistributed.com/2019/12/power-of-relationships.html

- Database Management System (DBMS): a software package designed to store and manage databases.
  - e.g. IBM DB2, Oracle, MS SQL Server, SAP Hana, Altibase, Postgres, MySQL, SQLite, .., and numerous New/NoSQLs

1.2 DBMS History & Big Guys

- See the supplementary notes at the end of this file

1.3 Files vs. DBMS

- Application must stage large datasets between main memory and secondary storage (e.g., buffering, page-oriented access, 64-bit addressing, etc.)
- Special codes for different queries
- Concurrency control & crash recovery
- Security and access control: flexible security policy

* What if emp and dept tables are stored as separate files and managed by file system and applications?

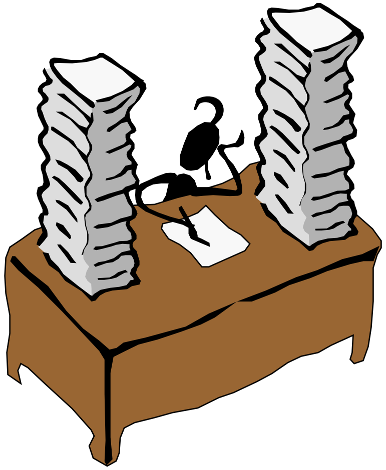
Very Large Data Bases

1.4 Why Use a DBMS?

- Data independence
- Efficient data access
- Data integrity and security
- Uniform data administration: DBA, no redundancy, tuning
- Concurrent access and crash recovery
- Reduced application development time
  - i.e. Application development productivity

Value of DBMS over File System!!
Why we have to cost for DBMS service?

Why Study Databases?



- Paradigm shift from computation to information
  - Computing-intensive vs. data-intensive
- Datasets increasing in diversity and volume.
  - Digital libraries, interactive video, Genome project, EOS project
  - New domains: social network (& analysis), twitter, Facebook, mobile applications (Kakao talk on top of SQLite)
  - The fourth paradigm of science (Jim Gray)
    - ✓ Empirical → Theoretical → Computational → Data-centric science
  - "Big data": 3V = Volume, Velocity, Variety
- DBMS encompasses most of CS
  - OS, computer architecture, languages, theory, AI, multimedia, logic

1.5 Data Models

- Data model: a collection of high-level concepts for describing data.
  - It hides many low-level storage details.

- A schema is a description of a particular collection of data, using the given data model.

- The relational model of data is the most widely used model today.
  - Main concept: relation, basically a table with rows and columns.
  - Every relation has a schema, which describes the columns, or fields.

Very Large Data Bases

1.5.1 Relational Data Model

- Schema
  - Relation name
  - Attribute (or field, column) and its type

Students(*sid:* `string`, *name:* `string`, *login:* `string`, *age:* `integer`, *gpa:* `real`)

- Relation instance
  - A set of records (or tuples, rows)

sid	*name*	*login*	*age*	*gpa*
53666	Jones	jones@cs	18	3.4
53688	Smith	smith@ee	18	3.2
53650	Smith	smith@math	19	3.8
53831	Madayan	madayan@music	11	1.8
53832	Guldu	guldu@music	12	2.0

- And, integrity constraints
  - E.g. unique sid, 10 < age < 40

1.5.2 Levels of Abstraction

- Many external schemas (or views)
  - Describe how users see the data.
- Single conceptual (or logical) schema
  - Defines logical structure
  - Describe the stored data in terms of data model
- Single physical schema
  - Describes the files and indexes used
  - Describe how the relations in conceptual schema are actually stored on secondary storage devices

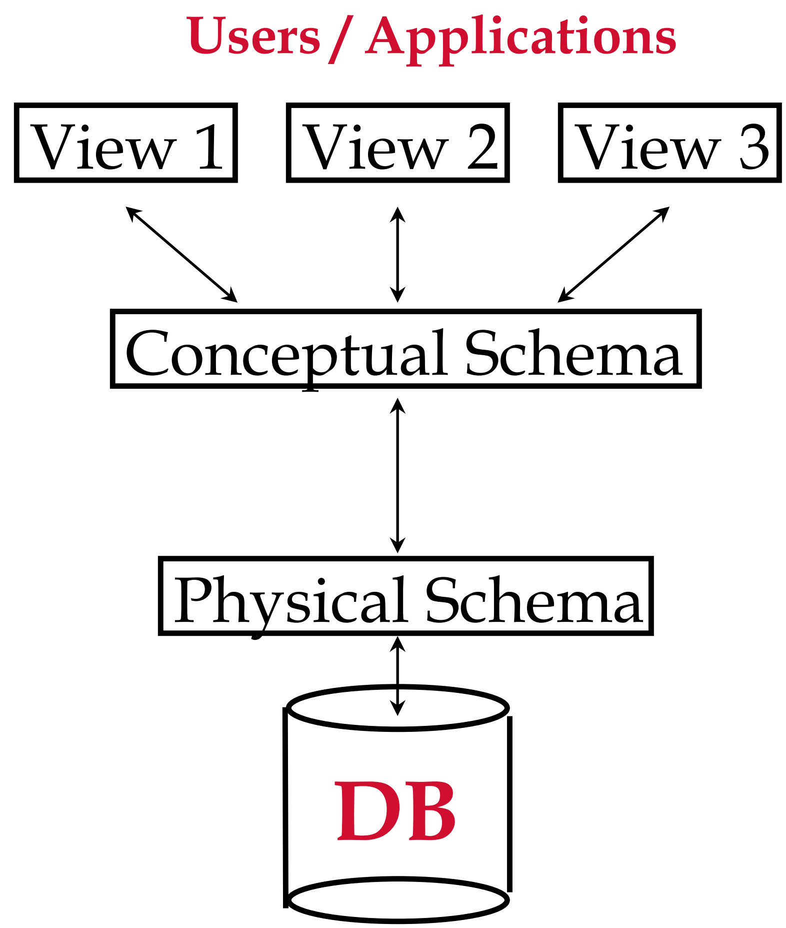


Figure 1.2 Levels of Data Abstraction

Example: University Database

- External schema (View):
  - *Course_info(cid:string,enrollment:integer)*
- Conceptual schema:
  - *Students(sid: string, name: string, login: string, age: integer, gpa:real)*
  - *Courses(cid: string, cname:string, credits:integer)*
  - *Enrolled(sid:string, cid:string, grade:string)*
- Physical schema:
  - Relations stored as unordered files.
  - Index on first column of Students.
  - Decisions about the physical schema are based on an understanding of how the data is physically accessed.

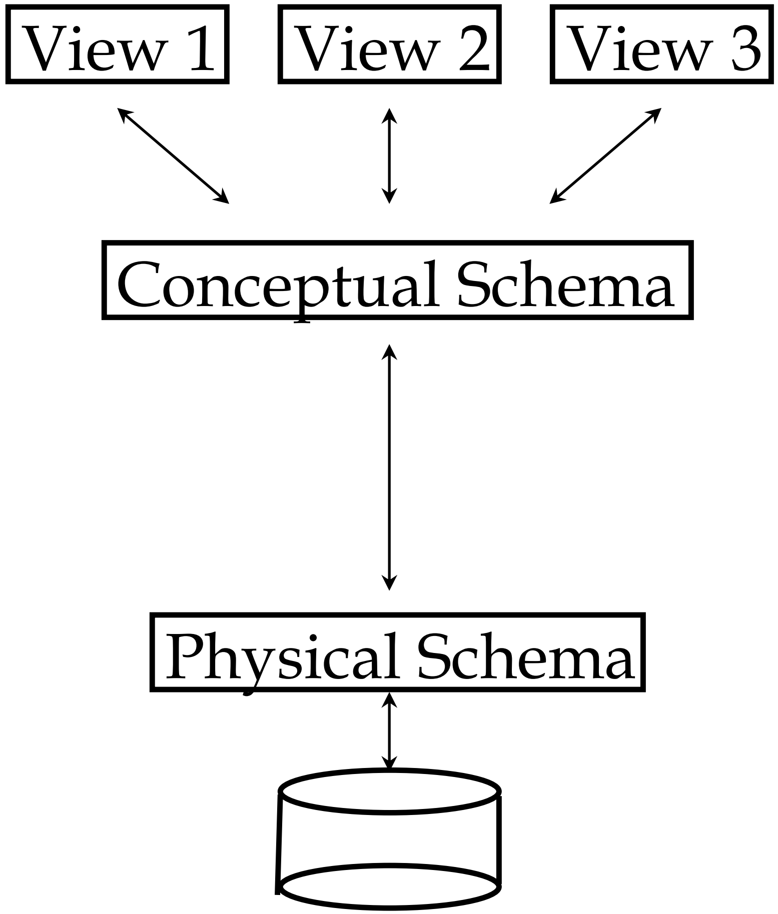


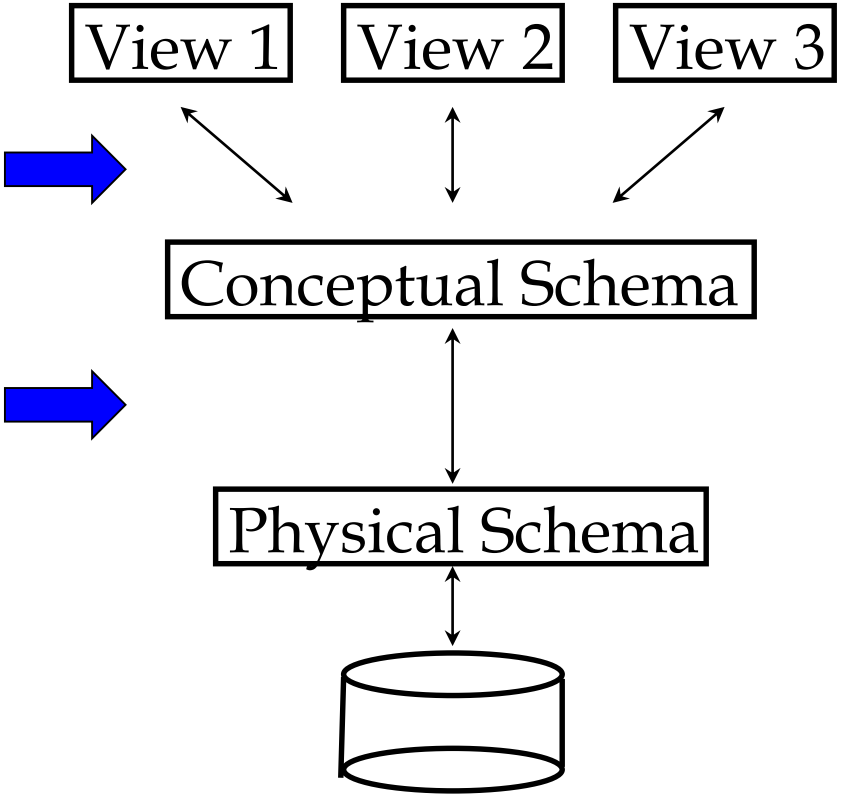
Figure 1.2 Levels of Data Abstraction

External/Logical/Physical Schema in RDBMS

- Data definition language (DDL) in SQL
  - External schema: Create view …
  - Logical schema: Create table …
  - Physical schema: Create index … ; create table … partitioning

Data Independence *

- Applications insulated from how data is structured and stored.
- **Logical data independence**: Protection from changes in *logical* structure of data.
- **Physical data independence**: Protection from changes in *physical* structure of data.



** One of the most important benefits of using a (Relational) DBMS!*

** Why is this so important for DBMS?*

1.6 Queries in a DBMS

- Queries = What user wants?
  - What is the name of the student with ID 123456?
  - What is the average salary of CS professors?
  - How many students are enrolled in CS564?
  - What percentage students received B or better in CS564?

- Query language: "WHAT"
  - Relational algebra & calculus (See Chap 4.)
  - Cf. DBMS query optimizer: "HOW" (See Chap 12 – 15)

  - Data sublanguage vs. host language (e.g. C, COBOL et)

1.7 Transactions

- "Life is full of transactions" - many actions of our daily life are transaction
  - e.g. Airline / train reservation, bank account, class registration, Kakao talk, Amazon, 배달의 민족, Coupang .....

```
BEGIN transaction;
update account set balance = balance - 10 where id = 1;
update account set balance = balance + 10 where id = 2;
COMMIT;
```

- Four properties of (computerized online) Transaction: ACID
  - **A**tomicity, **C**onsistency, **I**solation, **D**urability
  - Once you simply define your transactions with the simple syntax **"begin … commit"**, DBMS will do all the rest to guarantee ACID. (refer to ch1.example.sql script)
  - NOTE: Extremely difficult to achieve ACID with File System

- Think about the business processes in bank, if the bank is not computerized.
  - A banker is responsible for enforcing the biz. rule for money deposit
  - The banker is responsible for "all or nothing", concurrency, and durability

- In the computerized world, transaction processing systems (mostly, OLTP systems using DBMSs) is responsible for all the tasks including ACID.

1.7.1 Concurrency Control

- Concurrent execution of user programs is essential for good DBMS performance.
  - Because disk accesses are frequent, and relatively slow, it is important to keep the CPU humming by working on several user programs concurrently.
- Interleaving actions of different user programs can lead to inconsistency: e.g., check is cleared while account balance is being computed.
- DBMS ensures such problems don't arise: users can pretend they are using a single-user system.

```
BEGIN transaction;
update account set balance = balance - 10 where id = 1;
update account set balance = balance + 10 where id = 2;
COMMIT;
```

```
BEGIN transaction;
update account set balance = balance * 1.1;
COMMIT;
```

Transaction: An Execution of a DB Program

- Key concept is transaction, which is an atomic sequence of database actions (reads/writes).

```
BEGIN transaction;
update account set balance = balance - 10 where id = 1;
update account set balance = balance + 10 where id = 2;
COMMIT;
```

- Each transaction, executed completely, must leave DB in a consistent state if DB is consistent when the transaction begins.
  - Users can specify some simple integrity constraints on the data, and the DBMS will enforce these constraints.
  - Beyond this, the DBMS does not really understand the semantics of the data. (e.g., it does not understand how the interest on a bank account is computed).
  - Thus, ensuring that a transaction (run alone) preserves consistency is ultimately the user (i.e. application developer) responsibility!

Scheduling Concurrent Transactions

- DBMS ensures that execution of {T1, ... , Tn} is equivalent to some *serial* execution T1' ... Tn'.

T1:

```
BEGIN transaction;
update account set balance = balance - 10 where id = 1;
update account set balance = balance + 10 where id = 2;
COMMIT;
```

T2:

```
BEGIN transaction;
update account set balance = balance * 1.1;
COMMIT;
```

  - Before reading/writing an object, a transaction requests a lock on the object, and waits till the DBMS gives it the lock. All locks are released at the end of the transaction. (Strict 2PL locking protocol.)
  - Idea: If an action of Ti (say, writing X) affects Tj (which perhaps reads X), one of them, say Ti, will obtain the lock on X first and Tj is forced to wait until Ti completes; this effectively orders the transactions.
  - What if Tj already has a lock on Y and Ti later requests a lock on Y? (Deadlock!) Ti or Tj is aborted and restarted!

1.7.2 Ensuring Atomicity

- DBMS ensures *atomicity* (all-or-nothing property) even if system crashes in the middle of a Xact.

```
BEGIN transaction;
update account set balance = balance - 10 where id = 1;
update account set balance = balance + 10 where id = 2;
COMMIT;
```

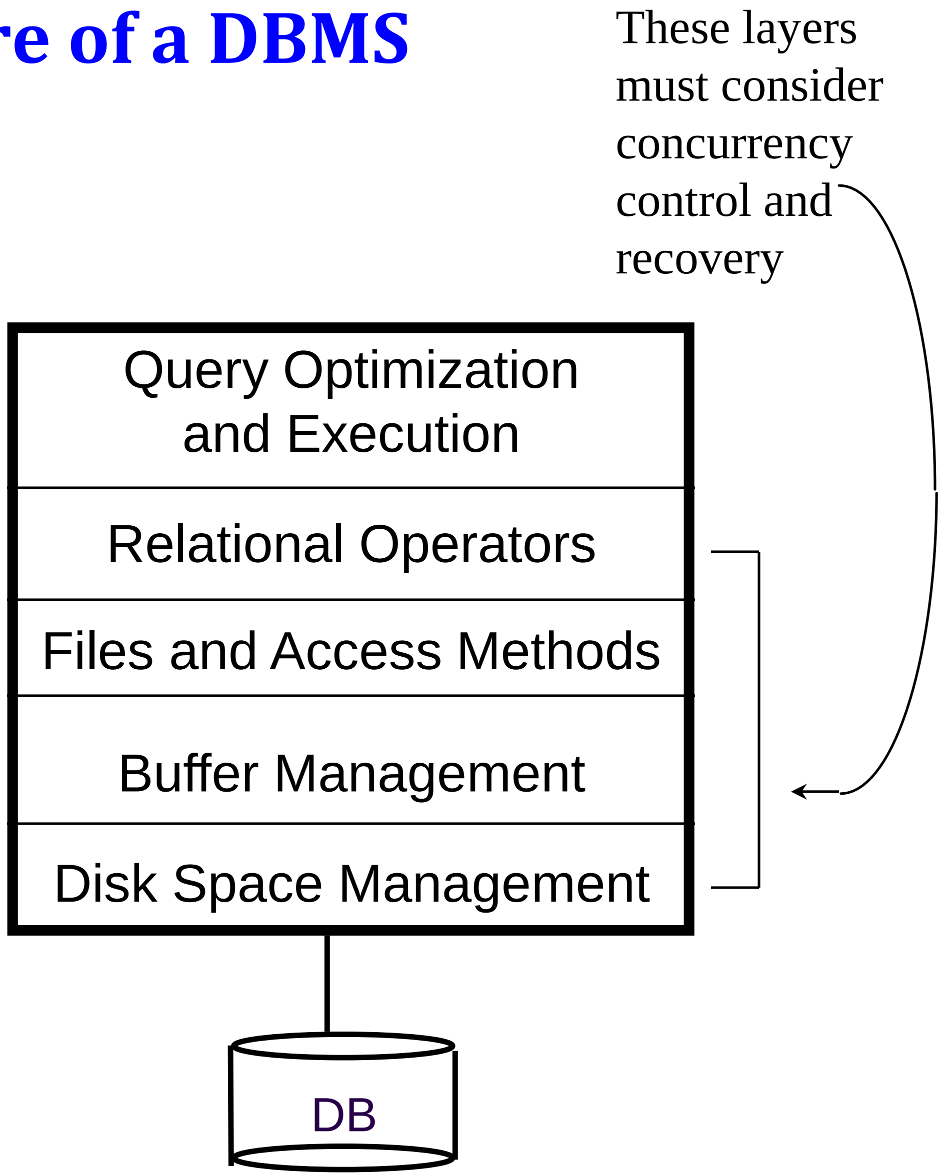
- Idea: Keep a *log* (history) of all actions carried out by the DBMS while executing a set of Xacts:
  - Before a change is made to the database, the corresponding log entry is forced to a safe location. (*WAL protocol*; OS support for this is often inadequate.)
  - After a crash, the effects of partially executed transactions are *undone* using the log. (Thanks to WAL, if log entry wasn't saved before the crash, corresponding change was not applied to database!)

The Log

- The following actions are recorded in the log:
  - *Ti writes an object*: the old value and the new value.
    - ✓ Log record must go to disk ***before*** the changed page!
  - *Ti commits/aborts*: a log record indicating this action.
- Log records chained together by Xact id, so it's easy to undo a specific Xact (e.g., to resolve a deadlock).
- Log is often *duplexed* and *archived* on "stable" storage.
- All log related activities (and in fact, all CC related activities such as lock/unlock, dealing with deadlocks etc.) are **handled transparently by the DBMS**.

1.8 Structure of a DBMS

- A typical DBMS has a layered architecture.
- The figure does not show the concurrency control and recovery components.
- This is one of several possible architectures; each system has its own variations.



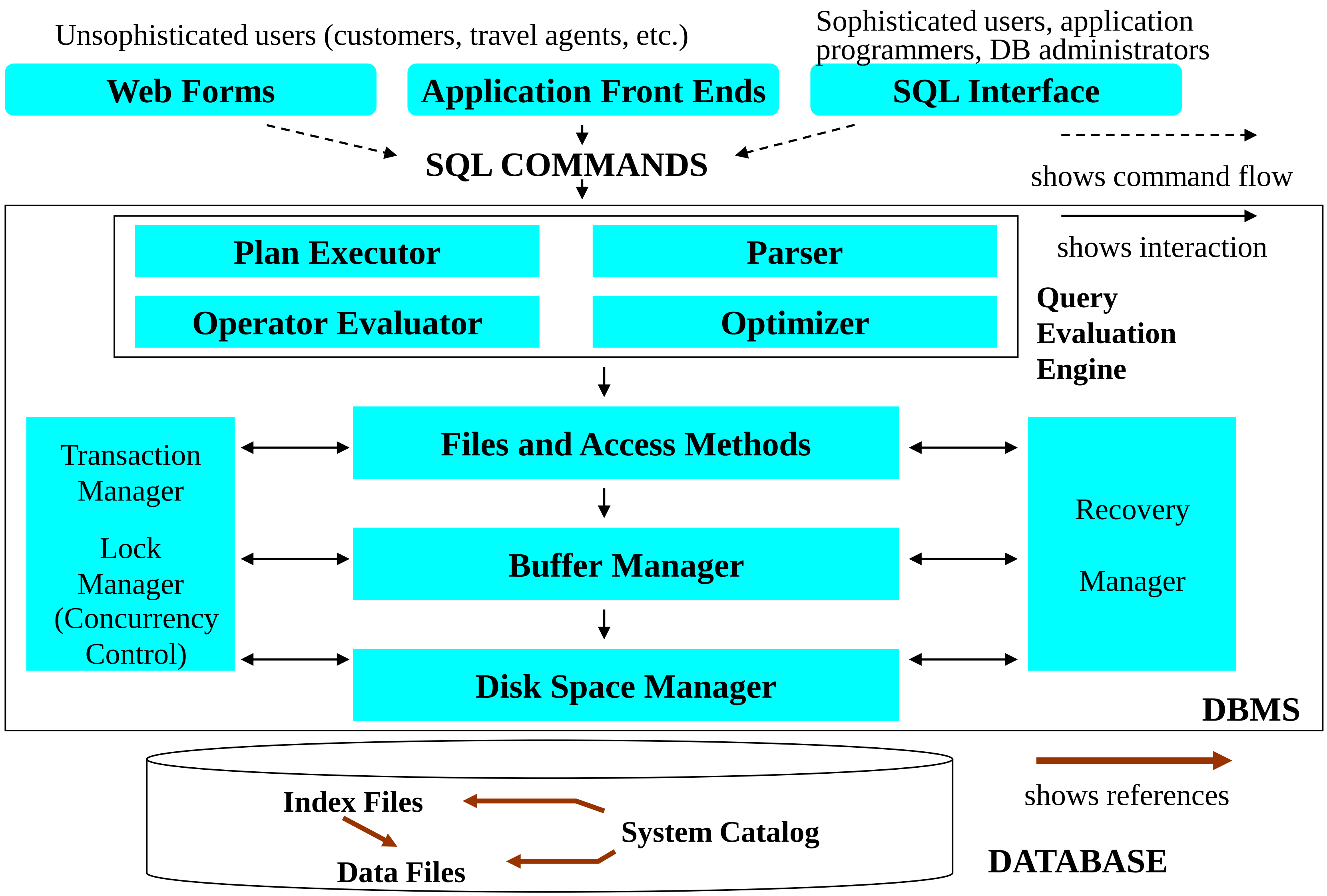
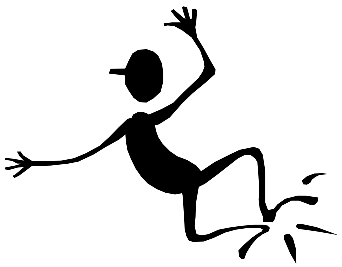


Figure 1.3 Anatomy of an RDBMS

1.9 Databases make these folks happy ...

- End users and DBMS vendors
  - Oracle Exadata: as hardware and server technology become commodity, Oracle expands its business down to server / storage

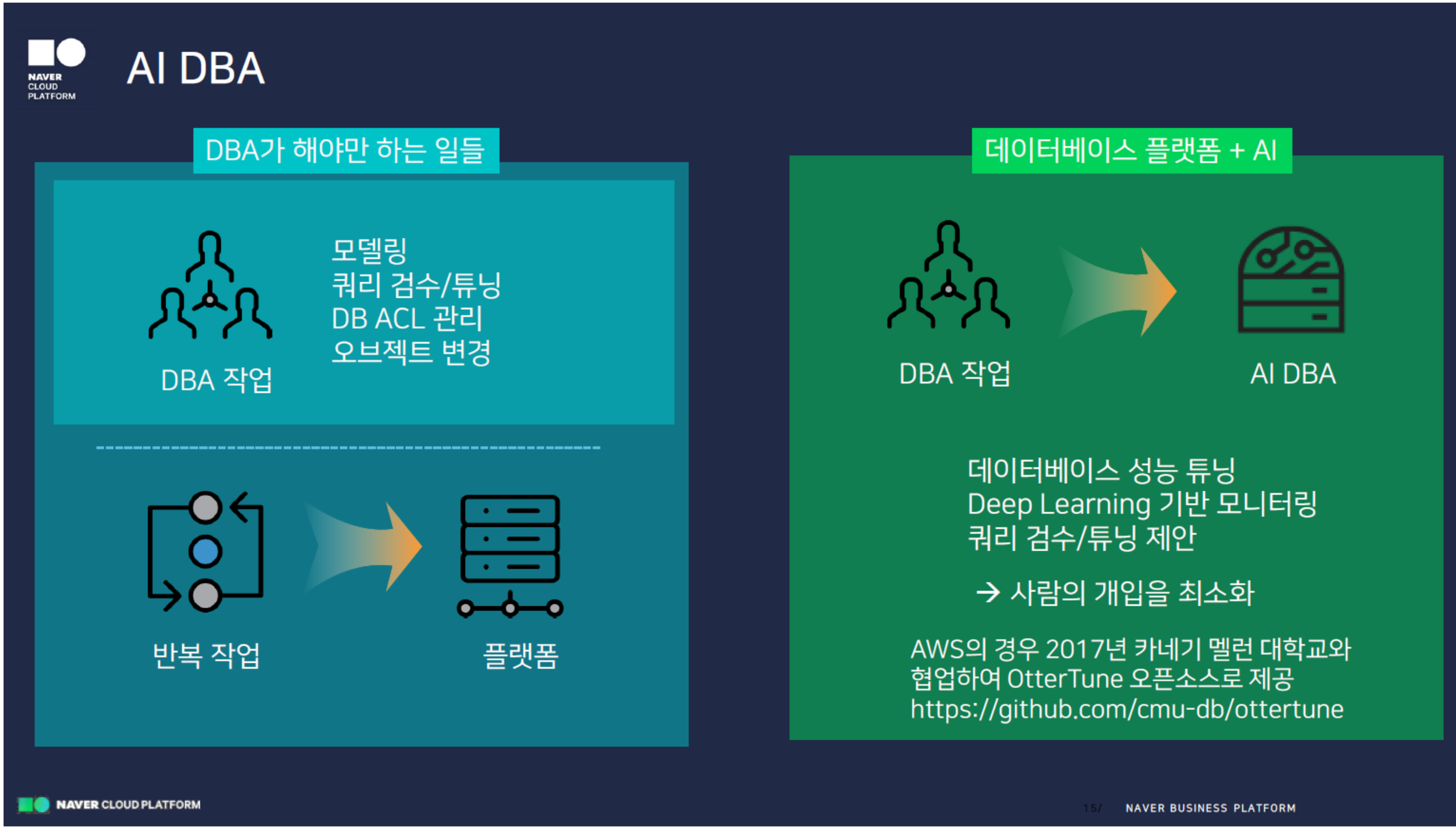


- DB application programmers
  - App developer using SQLite, Smart webmasters
- Database administrator (DBA)
  - Designs logical /physical schemas
  - Handles security and authorization
  - Data availability, crash recovery
  - Database tuning as needs evolve

You must understand how a DBMS works!

Very Large Data Bases

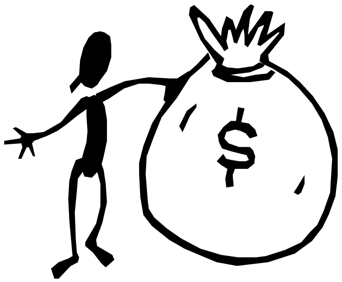
AI DBA @ NAVER


NAVER CLOUD PLATFORM
AI DBA
DBA가 해야만 하는 일들
모델링
쿼리 검수/튜닝
DB ACL 관리
오브젝트 변경
DBA 작업
반복 작업
플랫폼
데이터베이스 플랫폼 + AI
DBA 작업
AI DBA
데이터베이스 성능 튜닝
Deep Learning 기반 모니터링
쿼리 검수/튜닝 제안
→ 사람의 개입을 최소화
AWS의 경우 2017년 카네기 멜런 대학교와
협업하여 OtterTune 오픈소스로 제공
https://github.com/cmu-db/ottertune
NAVER CLOUD PLATFORM
NAVER BUSINESS PLATFORM

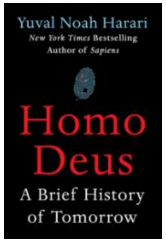
- http://headrush.typepad.com/creating_passionate_users/2006/03/how_to_be_an_ex.html

Summary

- DBMS used to maintain, query large datasets.
  - Recovery from system crashes, concurrent access, quick appl. development, data integrity and security.
- Levels of abstraction give data independence.
- A DBMS typically has a layered architecture.
- DBAs hold responsible jobs and are well-paid!(?)



- DBMS R&D is one of the broadest, most exciting areas in CS.
  - Big data and data scientist
- One of the most promising job in the next 10 years will be “data scientist” (i.e., dataist)?
  - Data as religion: saved by Algorithm
    - ✓ Dataism vs. {capitalism, religion, revolution}



Very Large Data Bases

Two Big Ideas of Relational DBMS

- **Declarative interfaces**
  - Apps specify *what* they want, **not** *how* to do it
    - ✓ E.g. "store a table with 2 columns", but not how to encode it on disk
    - ✓ E.g. "count records where column1 = 5", but not how to calculate it
- **Transactions**
  - Encapsulate multiple app actions into one *atomic* request (fails or succeeds as a whole)

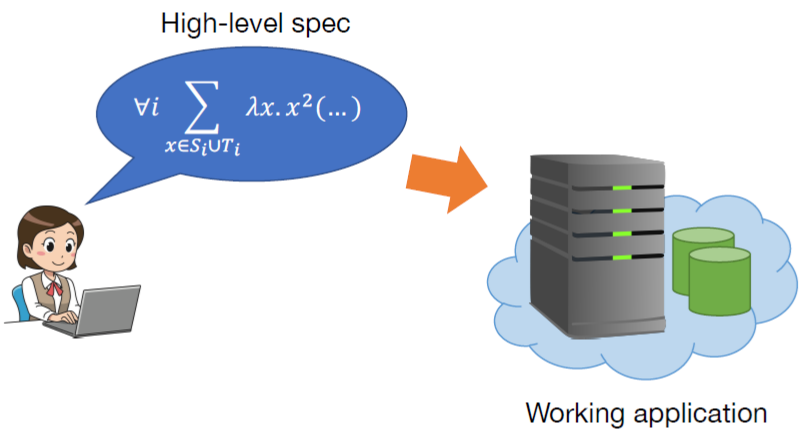
```
BEGIN transaction;
update account set balance = balance - 10 where id = 1;
update account set balance = balance + 10 where id = 2;
COMMIT;
```

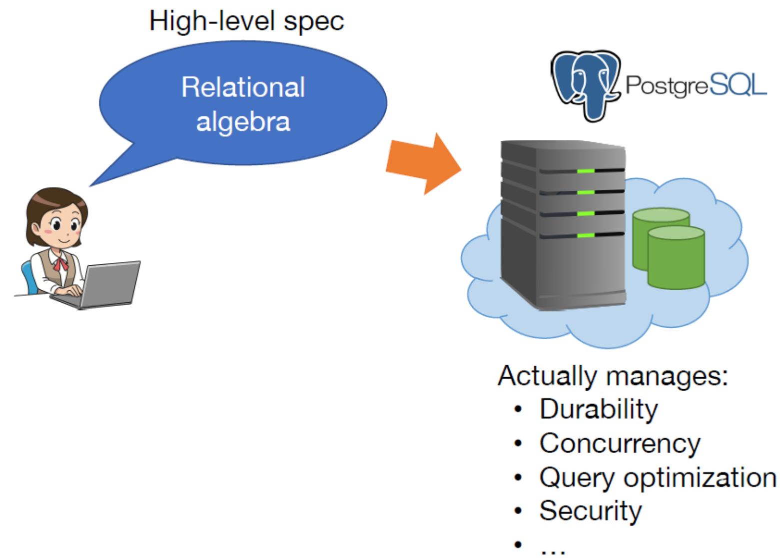
  - Concurrency models for multiple users
  - Clear interactions with failure recovery

User defines transaction, DBMS do the rest for ACID!

** source: Stanford CS245 by Matei Zaharia @http://web.stanford.edu/class/cs245/slides/01-Introduction.pdf

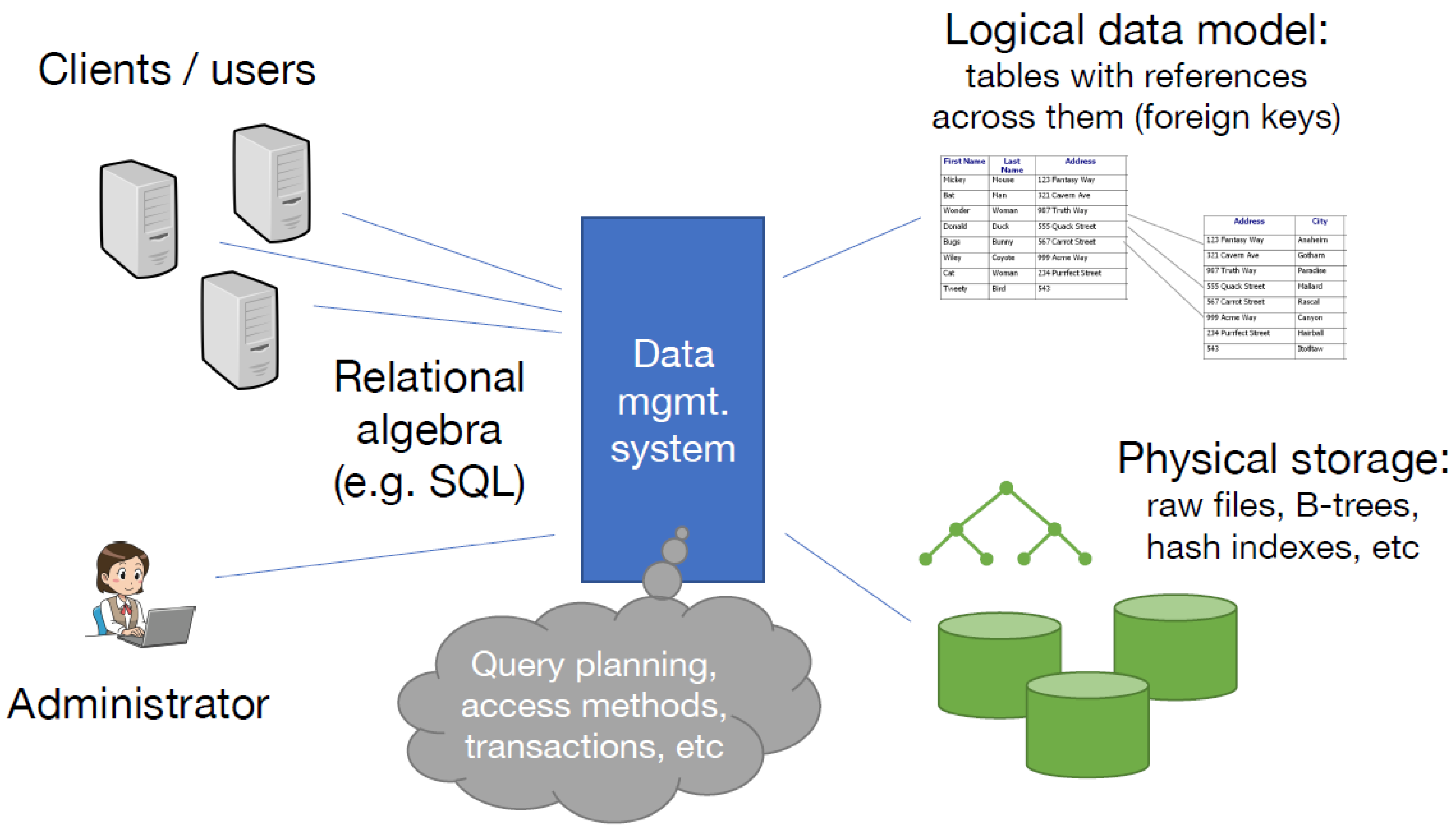
Data Programming: Dream vs. With Database





** source: Stanford CS245 by Matei Zaharia @http://web.stanford.edu/class/cs245/slides/01-Introduction.pdf

Key Ideas in Relational DBMS



First Name	Last Name	Address
Mickey	Mouse	123 Fantasy Way
Bat	Man	321 Cavern Ave
Wonder	Woman	987 Truth Way
Donald	Duck	555 Quack Street
Bugs	Bunny	567 Carrot Street
Wiley	Coyote	999 Acme Way
Cat	Woman	234 Purrfect Street
Tweety	Bird	543

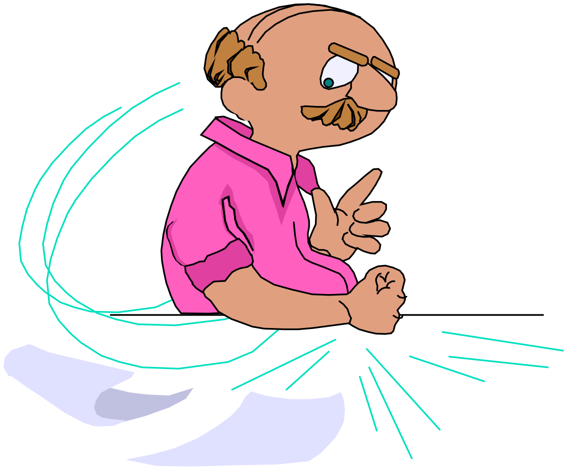
Address	City
123 Fantasy Way	Anaheim
321 Cavern Ave	Gotham
987 Truth Way	Paradise
555 Quack Street	Mallard
567 Carrot Street	Rascal
999 Acme Way	Canyon
234 Purrfect Street	Hairball
543	[illegible]

** source: Stanford CS245 by Matei Zaharia @http://web.stanford.edu/class/cs245/slides/01-Introduction.pdf

Very Large Data Bases

“I hear and I forget. I see and I remember. I do and I understand”

-- Chinese Proverb



Very Large Data Bases