Purpose of database system:

(*) Early systems were built on top of the file system leading to

→ Data redundancy & inconsistency: Duplication of data across formats.

→ Difficulty in accessing data: Each task needed custom programs.

→ Data isolation: Data stored in multiple formats made integration difficult.

→ Integrity problems: Constraints were embedded in code, making updates hard.

→ Atomicity issues: Partial updates could leave the system inconsistant.

→ Concurrency problem: Multiple users accessing data could cause inconsistencies

→ Security problem: Difficult to manage fine-grained access & control.

Data models

1. Relational models: stores data in tables (rows & columns)
3. Entity-Relationship model: Primarily used in database system.
3. object-based models: Combine object-oriented & relational concepts.
4. Semi-structured data model: XML
5. Older models: Network & hierarchical models.

Relational Model

→ Data is stored in tables (relations)

→ Example: Table containing rows & columns

View of Data

Architectural framework that separates the physical & logical structures of a database

Instances & schemas:

1. Logical schema: Describes the overall structure of the database.

2. Physical schema: Describes how data is physically stored.

3. Instances: The actual data in the database at a specific time.

Physical Data Independence:

→ The ability to change the physical storage without affecting the logical schema.

Data Definition Language (DDL)

→ Used to define database schemas (e.g creating tables)

→ Example

```
CREATE TABLE instructor(
    ID CHAR(5),
    name VARCHAR(20),
    salary number
);
```

Data manipulation language (DML)

→ Used for accessing & updation data

→ Declarative DML: Specifies what data is needed (easier to learn)

→ Procedural DML: Specifies how to retrieve the data.

SQL Query language:

```
SELECT name
FROM instructor
WHERE dep_name = 'comp. sci.';
```

Database Design

Logical Design: Deciding on the database schema.

Physical Design: Deciding on the physical layout of the database.

Storage manager: Manages the storage & retrieval of data, interacting with the OS file manager.

Query processor:
→ Interprets & compile DDL & DML queries.
→ Optimizes queries & generates low-level instructions for evaluation.

Transaction management
→ Ensures the consistency of the database even in the face of failures.
→ Concurrency control: Manages multiple transactions ~~of~~ to ensure data consistency

Database Architecture
→ Centralized Database: One or few cores. Shared memory.
→ Client – server database: Clients invoke services from a server.
→ Parallel Database: Shared memory or distributed architectures.
→ Distributed Database: Geographically distributed often heterogeneous schemas.

Database Users:

→ Naive Users: Use pre-written application programs.

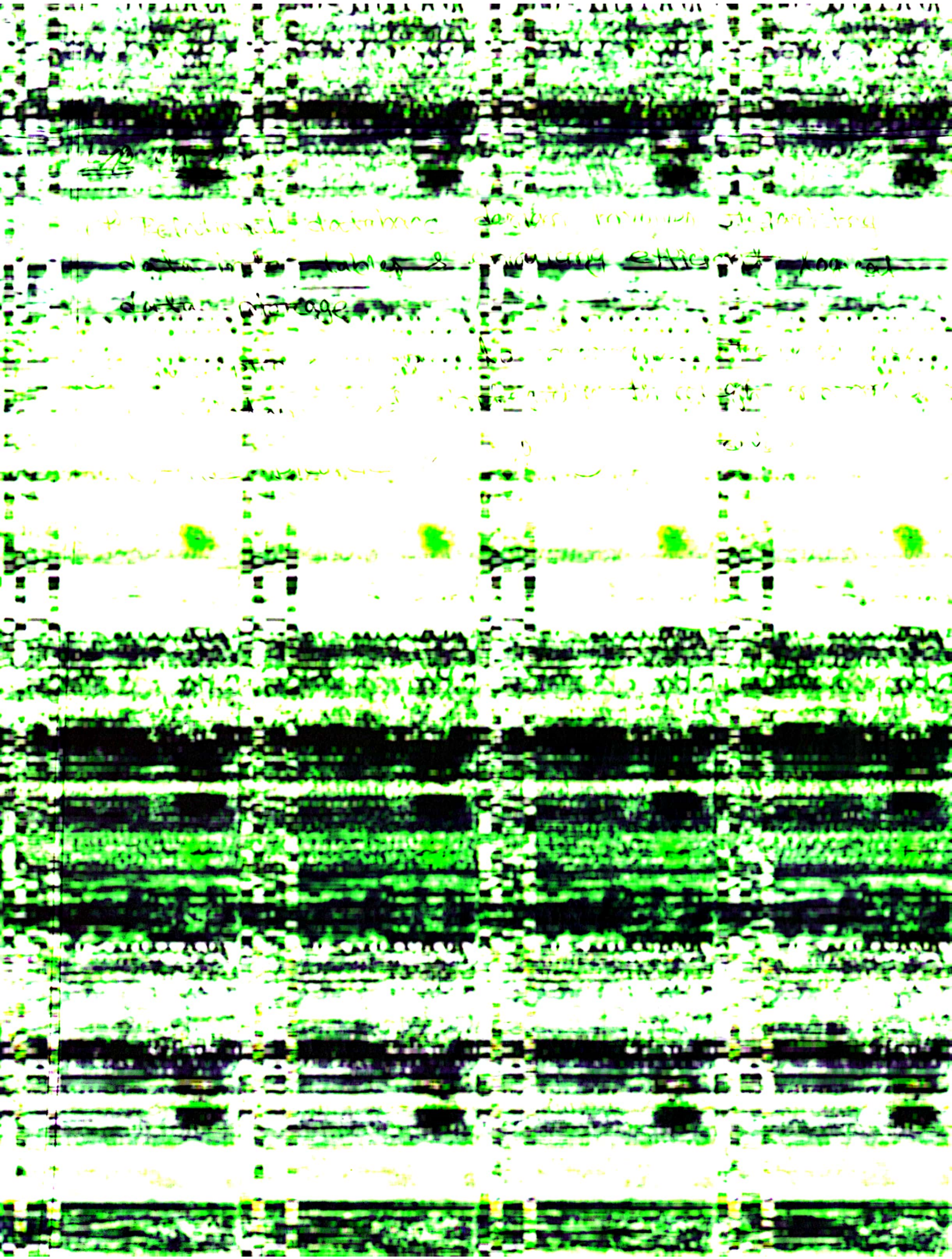
→ Application programmers: write application programs.

→ Sophisticated users: Use query languages or data analysis tools.

→ Specialized users: Develop unique applications outside traditional frameworks.

Database Administration (DBA)

Responsibilities include defining schemas, managing storage, ensuring database security & performing routine maintenance.



CamScanner

Database System

DBMS contains information about a particular enterprise

* Collection of interrelated data
* Set of programs to access the data.
* An environment that is both convenient & efficient to use.

Database systems are used to manage collections of data that are —

* Highly valuable
* Relatively large
* Accessed by multiple users & applications often at the same time.

A modern database system is a complex software system whose task is to —

* Manage a large complex collection of data
  Database touch all aspects of our lives.

Enterprise Information:

* Sales: Customers, products, purchases
* Accounting: Payments, receipts, assets
* Human Resources: Information about employees, salaries, payroll, taxes

Manufacturing:

Management of production, inventory, orders, supply chain.

Banking and Finance:

* Customer information, accounts, loans, and banking transactions
* Credit card transactions
* Finance: sales and purchases of financial instruments (e.g: stocks and bonds, storing real-time market data)

Universities: Registration, grades

Database Application Examples (cont.)

Airlines: Reservations, schedules.

Telecommunication: Records of calls, texts and data usage generating monthly bills, maintaining balances on prepaid calling cards.

Web based services:

*Online retailers: Order tracking, customized recommendations.

Document databases

Navigation systems

For maintaining the locations of varies places of interest along with the exact routes of roads train systems, buses etc.

Purpose of Database systems / Drawbacks of using file systems to store data.

In the early days, databases applications were built directly on top of file systems, which leads to:

Data redundancy and inconsistency:

* Data is stored in multiple file formats resulting in duplication of information in different files.

Difficulty in accessing data:

* Need to write a new program to carry out each new task.

Data isolation:

Multiple files and formats.

Integrity problems

* Integrity constraints (e.g: account balance > 5000) become 'buried' in program code rather than being state explicity.

* Hard to add new constraints or change existing ones.

Atomicity of Updates

* Failures may leave database is an inconsistant state with partial updates carried out.
* Example: Transfer of funds from one account to another should either complete or not happen at all.

Concurrent access by multiple users

* Concurrent access needed for performance
* Uncontrolled concurrent access can lead to inconsistancies

Example: Transfer of funds from one account to another should either complete or not happen at all.

Security problems:

Hard to provide user access to some but not all data.

University Database System Example

Data consists of information about:

* Students
* Instructors
* Classes

Application program examples:

* Add new students, instructors and courses.
* Register students for courses and generate class rosters.
* Assign grades to students, compute grade point averages (GPA) and generate.

View of Data

A major purpose of database system is to provide users with an abstract view of the data

Data Models :

* A collection of conceptual tools for describing data, data relationships, data semantics and consistency constraints.

Data abstraction:

* Hide the complexity of data structures to represent data in the database from users through several levels of data abstraction.

Data Models

A collection of tools for describing

* Data
* Data relationship
* Data semantics
* Data constraints

Relational Model:

Entity-relationship data model (mainly for database design)

"Create database UAP"

For drop

```
DROP TABLE [master].[dbo].[department_old]
USE UAP
```

For creating table

```
Create table department_old
(
dept_name char(20),
building varchar(10),
budget money,
)
```

For insert

```
INSERT into[dbo].[department_old] values
('CSE', 'RH' 13000)
INSERT into [dbo].[department_dd] values
('Null','RH', 13000) otr
INSERT into department_old values
```

For select

```
select * from [dbo].[department_old]
```

For create table

```
create table department
(
dept_name char(20),
building varchar(10),
budget int,
constraint d_pk primary key (dept_name)
)
```

For delete

```
delete from department
where dept_name = "Null"
delete from department
where dept_name is null
delete from department
where building NULL
```

Defination

Schema: Overall design or plan of a database

It shows how data is organized & connected but it doesn't include actual data itself.

Types

1. Physical schema: Explain how data stored physically

2. logical schema: Shows the logical structure of the data such as tables, columns and relationship between them

3. Sub-schema: Describes the different ways users can see the data.

Logical schema:

Defination: The logical schema is the abstract or high level design of the database

concept: It focous on, what data is stored & how its related without considering how its stored physically.

Example: A table with student name, addresses & courses in part of the logical schema.

*SQL is used for managing and ~~querying~~ querying relational database.

(*) Data Manipulation Language (DML): Allowing querying, inserting, deleting and modifying tuples in the database.

(*) Data Definition Language (DDL): Used to define schemas, integrity constraints, views & control transactions.

(*) Integrity constraints: Ensures data accuracy (primary key, foreign key, not null constraints)

(*) DDL – Data Defination Language

- Used for specifying relations (tables), including attributes & types & integrity constraints.

Example:

```
Create table instructor(
    ID char(5),
    name varchar(20),
    dept_name varchar(20),
    salary numeric(8,2)
    );
```

Domain type

- char(n) : Fixed length character string
- varchar(n) : variable length character string.
- int, smallint : Integer values.
- numeric (p,d) : Fixed point numbers.
- real, double precision : Floating point numbers.

Integrity constraints

- Primary key — Uniquely identifies rows.
- Foreign key — References another table to maintain relationships.
- Not null : Ensures a column cannot have null values.

SQL Queries :

1. select clause : Used to specify the columns to retrieve

```
select name from instructor;
```

- use distinct to remove dublicates and * to select all column.

2. Where clause: Applies conditions to filter results.

Example:

```
select name from instructor where
                    dept_name = 'comp. sci.';
```

3. From clause: Specifies the tables involved in the query.

Example: select * from instructor

4. Join operators: Combining two or more tables based on a related column.

5. Rename operation: SQL allows renaming relations and attribute using the as clause.

example

```
select distinct table name
from instructor as instructor table,
instructor as salary
where table salary) s. salary and s. dept_name
                    = 'comp. sci';
```

④ String operation:

- The like operator is used for pattern matching
  - '%' : matches any substring
  - '_' : " a single character.

example:

```
select name from instructor where
        name like '%dar%';
```

Set operation

① Union: Combine results from two queries.

Example:

```
(select course_id from section where
    sem = 'Fall' and year = 2017)
union
(select course_id from section where
    sem = 'Spring' & year = 2018);
```

② Intersects: Returns common results between two queries.

Example:

```
(select course_id from section where sem='Fall'
    and year = 2017)
intersects
(select course_id from section where sem = 'Spring' &
                                    year = 2018);
```

• Except : Return the difference between two queries.

Exm:

```
(select course_id from section where sem = 'Fall' and
        year = 2017)
except
(select course_id from section where sem = 'Spring' and
        year = 2018);
```

Null values :

- SQL allows null values, representing unknown or missing data.
- use null or is not null to check for nulls.

Exm: select name from instructor where salary is null.

Aggregate function

- Operate on sets of values to return a single value.
  - avg() → calculates average.
  - min() → finds the minimum
  - max() → " " maximum
  - sum() → sum the values.
  - count() → counts the number of values.

example :

```
select avg(salary) from instructor where dept_name =
                    'comp. sci.';
```

Group by & Having

- Group by : used to group rows sharing a value.
- Having clause : Applies conditions to groups after aggregation.

example:

```
select dept_name, avg(salary) as avg_salary
    from instructor grouped by dept_name
    having avg(salary) > 42000;
```

Nested subqueries:

- SQL allows queries with in queries(subqueries)

Exm:

```
select name from instructor where salary >
    (select avg(salary) from instructor);
```

Group by : Group rows sharing a value

```
select dept_name, avg(salary) as avg_salary from
    instructor group by dept_name;
```

Having : Filters groups after aggregation:

```
select dept_name, avg(salary) as avg_salary
from instructor
group by dept_name
having avg(salary) > 42000;
```

Subqueries in the from clause:

- SQL allows the use of subquery expressions in the from clause, treating the results as a temporary relation.

Example:

```
select dept_name, avg_salary
from (select dept_name, avg(salary) as avg_salary
      from instructor
      group by dept_name)
where avg_salary > 42000;
```

With clause

- with clause in SQL allows the creation of temporary relations that can be reused within a query.

exm:

```
with max_budget(value) as (
        select max(budget)
        from department
        )
select department.name
from department, max_budget
where department.budget = max_budget.valu
```

Complex queries using with clause

- with clause can be used to break complex queries into smaller, more manageable parts.

example:

```
with dept_total(dept_name, value) as (
    select dept_name, sum(salary)
    from instructor
    group by dept_name
    );
dept_total_avg(value) as (
    select avg(value)
    from dept_total
)
select dept_name
from dept_total, dept_total_avg
where dept_total.value > dept_total_avg.
                         value;
```

Scalar subquery

A scalar subquery is a subquery that returns a single value, typically used where a single value is expected like in the select, where or from clause.

example:

```
select dept_name,
    (select count(*)
    from instructor
    where department.dept_name = instructor.dept_name)
    as num_instructors
    from department;
```

Modification of the database:

- Insert: Add data to a table

example: insert into instructor values ('10211', 'Smith', 'Biology', 66000);

- Delete: Delete data from a table

example: delete from instructor where dept_name = 'Finance';

- Update: Modify existing data.

example: update instructor set salary = salary * 1.05 where salary < 70000;

Case statement:

- conditional logic inside SQL queries.

example:

```
update instructor
set salary = case
    when salary <= 10000 than salary * 1.05
    else salary * 1.03
    end;
```

Structure of relational database:

* A relation is essentially a table in a database, with attributes (columns) and tuples (rows)
* The domain of an attributes refers to the set of possible values the attribute can take.
* Each value in a relational database must be atomic (invisible)
* Special values such as null can exist, representing unknown or inapplicable data, but they introduce complications in operations.

Relations are unordered

→ The order of tuples in a relation is irrelevent

→ Relations can be stored in any arbitrary order

Database schema

- A database schema defines the logical structure of the database, while a database instance is the data at a specific point in time

Example: instructor (ID, name, dept-name, salary)

Key

- A superkey is a set of attributes whose values uniquely identifies a tuple

Example: {ID} or {ID, name} are superkeys of instructor

- A candidate key is a minimal superkey
  Example: {ID} is a candidate key for the instructor.
- One candidate key is designated as the primary key.
- A foreign key is one relation references the primary key in another.
  Example: dept_name in instructor is a foreign key referencing department.

Schema diagram :

- A schema diagram visually represents the relationships between different tables in a database.

For example: A university database might show relations like instructor, department & teaches.

Relational Query language

- Procedural vs Non-procedural (Declarative) language: Procedural requires specifying how to obtain results, whereas declarative focuses on what results are needed.
- Relational algebra, tuple relational calculus and domain relational calculus are equivalent in expression power.
- Relational algebra is a procedural language used for querying relational database

Relational Algebra

- Relational algebra consists of several basic operations.

1. Select (σ): Filters rows based on a predicate

Example: σ dept_name = "physics" (instructor) selects instructors in the physics department.

2. Project (Π): Retrieves specified columns, eliminating duplicates.

Example: Π ID, name, salary (instructor) returns only the ID, name, and salary of instructors.

3. Union (∪): Combines two relations with the same attributes.

4. Set difference (−): Finds tuples in one relation but not another.

5. Cartesian product (×): Combines tuples from two relations.

6. Rename (ρ): Assigns a new name to a relation.

Cartesian product

- The cartesian product operation generates all possible combinations of tuples from two relation.

Example: instructor X teaches creates a relation with every possible instructor-teaches combinatio

Join operation:

- The join operation combines relations based on a condition.

Example: σ instructor.ID = teaches.id

(instructor X teaches) returns only the rows where instructor.id matches teaches.id.

Union, intersection & set difference

- Union combines two relations
- Intersection finds common tuple between two relation.
- Set difference finds tuple that are in one relation but not in the other.

Assignment and Rename

- Assignment ($\leftarrow$): Useful for breaking complex queries into smaller parts.
- Rename (ρ): Assigns names to intermediate results in relational algebra expressions.

Equivalent Queries:

- Multiple expressions in relational algebra can yield the same result, even if they are written differently.

Eg:

(1) σ dept_name = "Physics" $\wedge$ salary > 90000 (instructor)

(2) σ dept_name = "Physics" (σ salary > 90000 (instructor)).

(3) Both yield the same result but use different compositions of operations.