CMPT 506

Review of Fundamental Database Concepts

Dr. Abdelkarim Erradi

Department of Computer Science & Engineering

QU

Outline

1. Why DBMS?
2. ACID = Gold Standard For RDBMS Design
3. DBMS Architecture
4. Conceptual Data Modeling
5. Review of Normalization (self-study)

Acknowledgment

Some slides adapted from Dr Shivnath Babu advanced database course

Why DBMS?

Back

What is a database system?

From Oxford Dictionary:

- Database: an organized body of related information
- DataBase Management System (DBMS): a software system for providing **efficient**, **convenient**, and **safe storage** of and **multi-user access** to, possibly **massive**, amounts of **persistent** data.

DataBase Management System (DBMS)

High-level
Query *Q*

Answer

DBMS

- Translates Q into best **execution plan** for **current conditions**,
- Runs plan

Keeps data safe and correct despite failures, concurrent updates, online processing, etc.

Data

Example Queries: At a Company

Query 1: Is there an employee named “Nemo”?
Query 2: What is “Nemo’s” salary?
Query 3: How many departments are there in the company?
Query 4: What is the name of “Nemo’s” department?
Query 5: How many employees are there in the “Accounts” department?

Employee

ID	Name	DeptID	Salary	…
10	Nemo	12	120K	…
20	Ali	156	79K	…
40	Fatima	89	76K	…
52	Saleh	34	85K	…
…	…	…	…	…

Department

ID	Name	…
12	IT	…
34	Accounts	…
89	HR	…
156	Marketing	…
…	…	…

Example: Store that Sells Cars

Owners of Honda Accord who are <= 23 years old

Make	Model	OwnerID	ID	Name	Age
Honda	Accord	12	12	Nemo	22
Honda	Accord	156	156	Ali	21

Join (Cars.OwnerID = Owners.ID)

Filter (Make = Honda and Model = Accord)

Filter (Age <= 23)

Cars

Make	Model	OwnerID
Honda	Accord	12
Toyota	Camry	34
Mini	Cooper	89
Honda	Accord	156
…	…	…

Owners

ID	Name	Age
12	Nemo	22
34	Fatima	42
89	Saleh	36
156	Ali	21
…	…	…

History

Pre-relational DB (70)

- Data stored and mananed in files

Relational Database Systems (80)

- No redundancy in data storage
- Multiuser operation and high performance
- ACID Properties

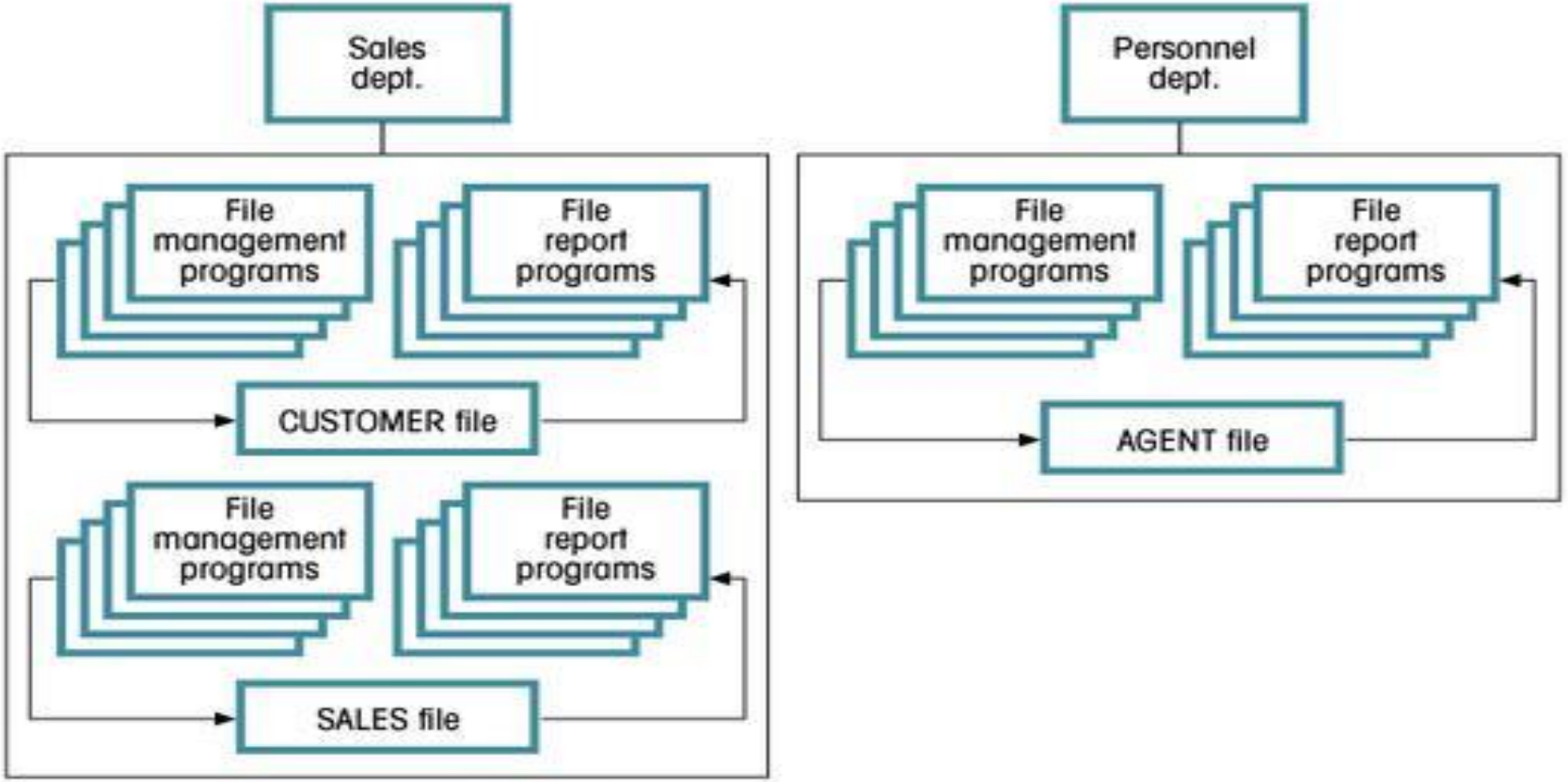
Post-relational Databases (90)

- Object-oriented Databases
- Distributed databases
- Datawarehouses

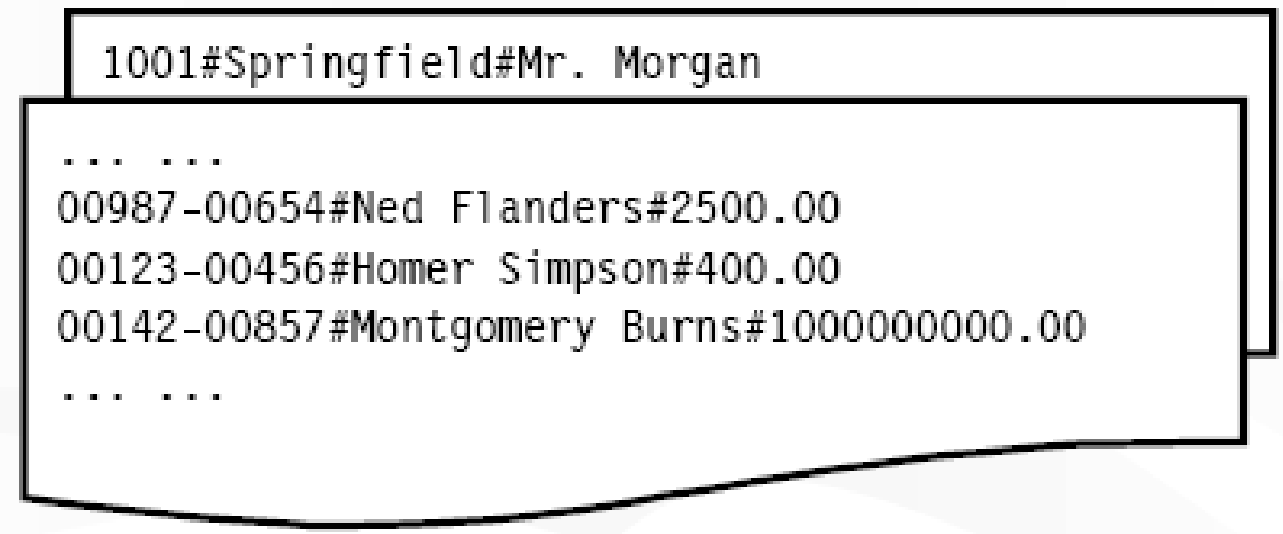
NoSQL Movement (21st Centry)

=> Latest Trends: NoSQL, NewSQL and Cloud Data Services

A Simple File System



Data file physical organization



- ASCII file
- Accounts/branches separated by newlines
- Fields separated by #'s

Query: What's the balance in Homer Simpson's account?

Answering Query using Imperative Approach

- What's the balance in Homer Simpson's account?

  => A simple script:

  – Scan through the accounts file
  – Look for the line containing "Homer Simpson"
  – Print out the balance
- Query processing tricks when having thousands accounts:
  – Cluster accounts: Those owned by "A..." go into file A; those owned by "B..." go into file B; etc.
  – Keep the accounts sorted by owner name
  – Hash accounts according to owner name
  – And the list goes on…

Observations

- To write correct code, application programmers need to know **how data is organized physically** (e.g., which indexes exist)
  - Burden on programmer to figure out right tricks to retrieve the data fast
- Tons of tricks (not only in storage and query processing, but also in concurrency control, recovery, etc.)
- Different tricks may work better in different usage scenarios
- Same tricks get used over and over again in different applications

Drawbacks of using file systems to store data

- To retrieve data from a file system, extensive programming is often needed - both *what* and *how* are programmer's responsibility
- *Ad hoc queries* require programming
- Each file must have its own file-management program to create the file structure, add data to file, delete data from it, modify it and list its contents
- All data access programs are subject to change when the file structure changes (e.g., a field is deleted or its position is changed)
  - ***Structural dependency***
- Even a change in the data type of a field (e.g., integer to real) requires all data access programs to change
  - ***Data dependency***

Drawbacks of using file systems to store data

- Data redundancy and inconsistency
  - Multiple file formats, 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 => difficulty to make joins
- Integrity problems
  - Integrity constraints (e.g. account balance > 0) become part of program code
  - Hard to add new constraints or change existing ones

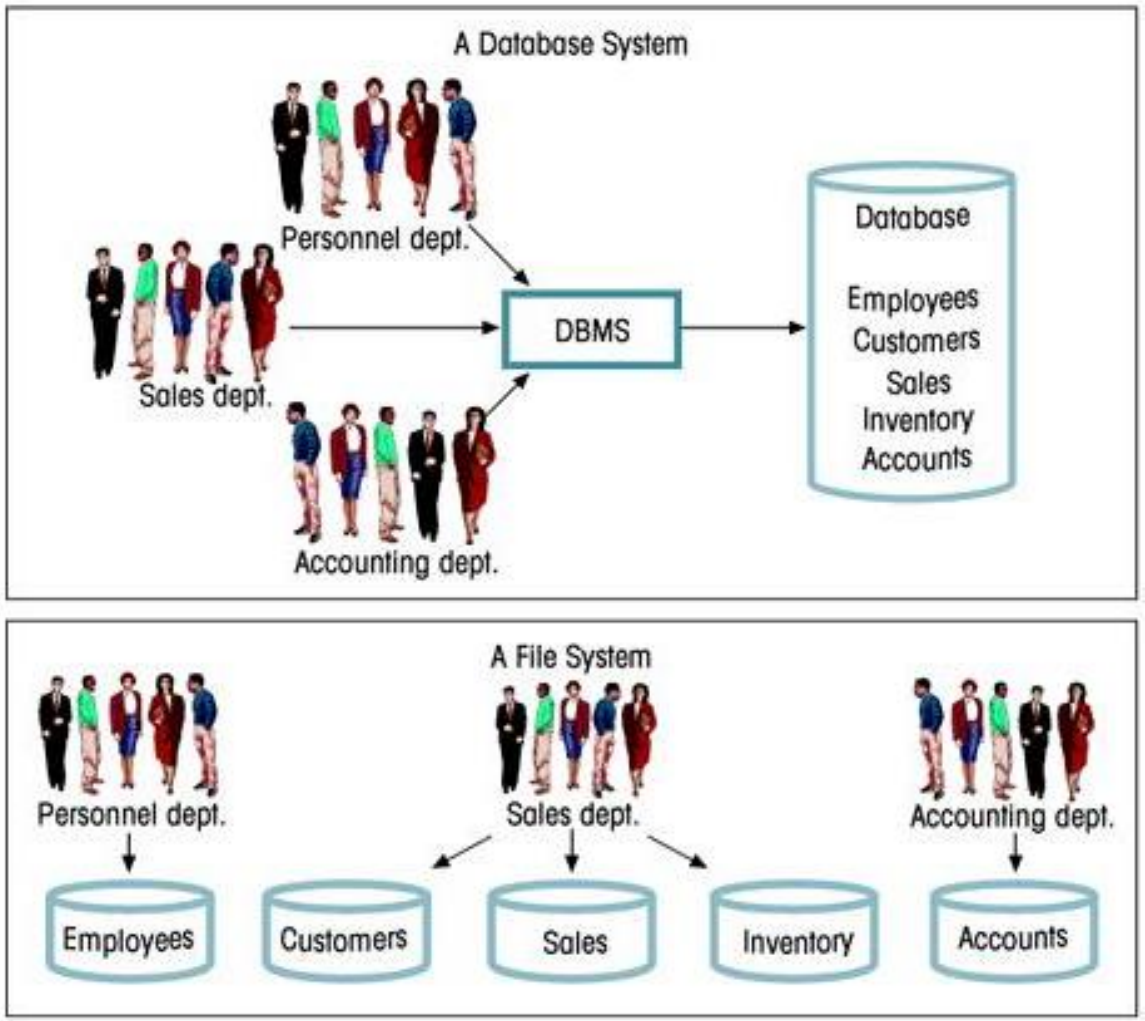
Drawbacks of using file systems (cont.)

- Hard to support **Atomicity** of updates
  - Failures may leave database in an inconsistent state with partial updates carried out
  - e.g. transfer of funds from one account to another should either complete or not happen at all
- Hard to support **Concurrent access** by multiple users
  - Concurrent accessed needed for performance
  - Uncontrolled concurrent accesses can lead to inconsistencies
    - E.g. two people reading a balance and updating it at the same time
- Security problems

=> Database systems offer solutions to all the above problems

DBMS vs. File Systems?

- Database consists of logically related data **stored in a single repository**
- Provides advantages over file system management approach
  - Eliminates inconsistency, data anomalies, data dependency, and structural dependency problems
  - Stores data structures, relationships, and access paths



DBMS two important questions

- What's the right interface to be used by the programmer?
  - **Data model**: Used to specify how data are conceptually structured
  - **Query language**: Used to specify data processing/management tasks

- How DBMS support this interface efficiently?
  - **Physical data organization**: Store and index data in smart ways to speed up access
  - **Query processing and optimization**: Figure out the most efficient method to carry out a given task

The birth of DBMS

The relational revolution (1970's)

- **A simple data model**: data is stored in relations (tables)
- **A declarative query language**: SQL

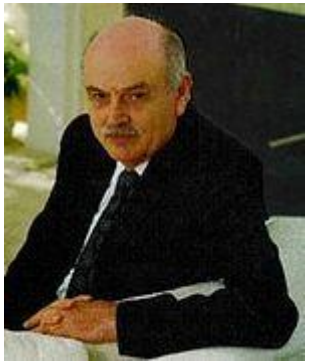
```
SELECT Account.owner
       FROM Account join Branch
              on Account.branch_id = Branch.branch_id
       WHERE Account.balance = 0
       AND Branch.location = 'Springfield';
```

- Programmer specifies **what** answers a query should return, but **not how** the query is executed
- DBMS picks the best execution strategy based on availability of indexes, data/workload characteristics, etc.

=> Provides physical data independence

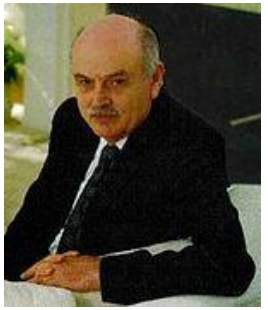
Goal of Data Management and Storage

- "Future users of large data banks must be protected from **having to know how the data is organized** in the machine (the internal representation)." - E. F. Codd, 1970



Physical data independence

- Applications should not need to worry about how data is physically structured and stored
- Applications should work with a **logical data** model and **declarative query language**
  - **Specify what you want, not how to get it**
  - Leave the implementation details and optimization to DBMS
- This principle if the most important reason behind the success of DBMS today
  - Edgar Frank **Codd** got a Turing Award for this



DB Levels of Abstraction

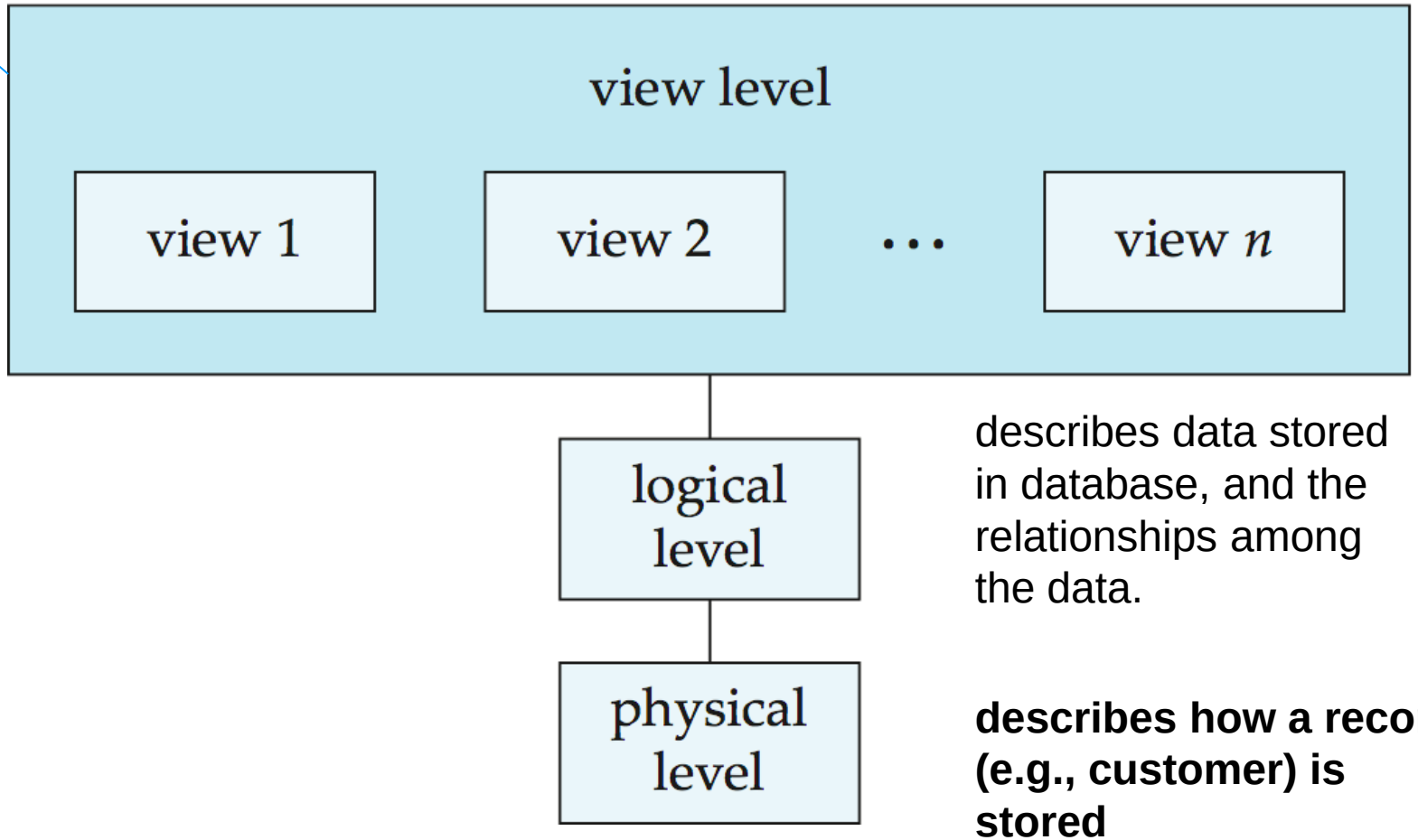
- **Physical level**: describes how a record (e.g., customer) is stored
- **Logical level**: describes data stored in database, and the relationships among the data.

```
type customer = record
                    name : string;
                    street : string;
                    city : integer;
end;
```

- **View level**: hide details of the underlying tables (e.g., hide employee's salary for security purposes)

DB Levels of Abstraction

Views hide details of the underlying tables (e.g., hide employee's salary for security purposes)



describes data stored in database, and the relationships among the data.

describes how a record (e.g., customer) is stored

DBMS Functions

- **Data dictionary**: stores data relationships to be consulted by all programs that access the database; automatically records changes

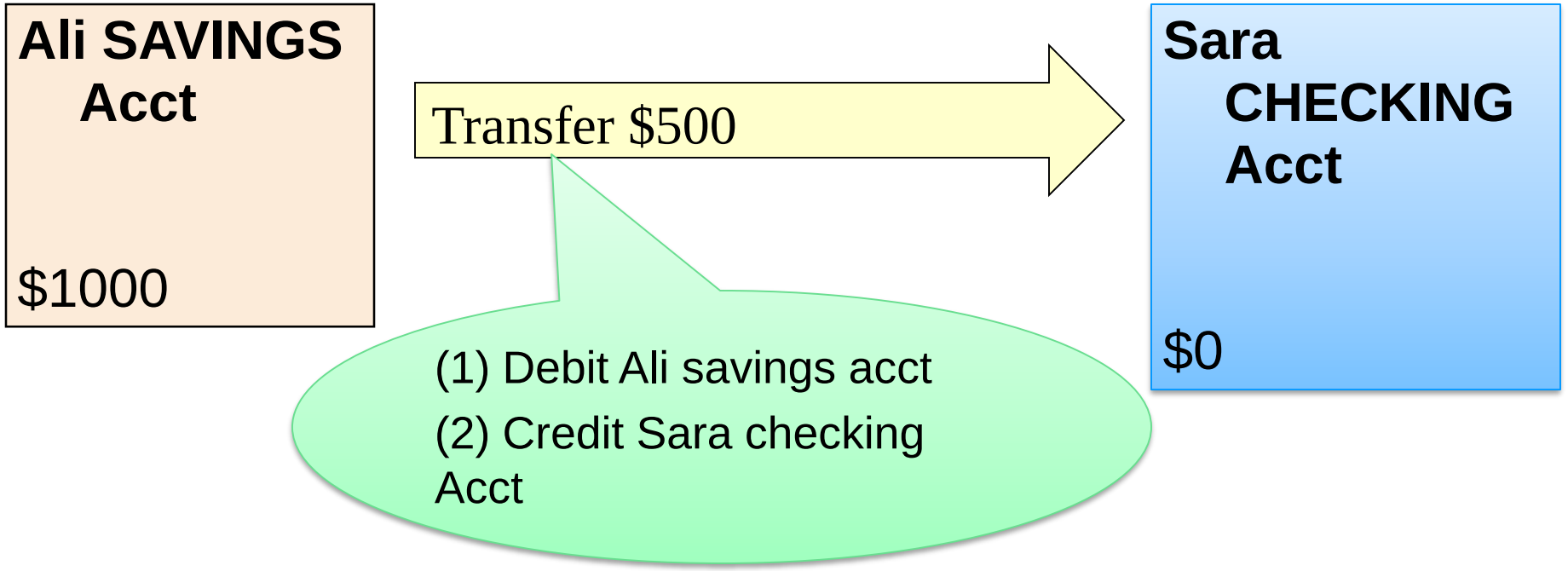
  - Structural and data dependencies are eliminated

- Interaction with the file manager: takes care of structures for data storage
- Multi-user access control
- Data integrity
- Can be used as ***integration mechanism between applications*** (shared database used by multiple applications)
- Security: enforces data access policies
- Backup and recovery
- Data access through many interfaces such as JDBC, ADO.Net

ACID = Gold Standard For RDBMS Design

Back

A transaction is a sequence of operations that must be executed as a whole



Either both (1) and (2) happen or neither!

Every DB action takes place inside a transaction

RDMS ACID properties

- **Atomicity**: Everything in a transaction succeeds or the entire transaction is rolled back (All or Nothing)
- **Consistency**: data inserts/updates/deletes do not violate any defined rules such as constraints, triggers
- **Isolation**: Transactions cannot interfere with each other => The updates of a transaction must not be made visible to other transactions until it is committed
- **Durability**: Results from completed transactions survive failures (e.g., power loss, crashes, or errors)

Example of consistency issues caused by concurrent updates

- **Example to illustrate consistency issues that can be introduced by concurrent updates:**

```
Get account balance from database;
If balance > amount of withdrawal then
   balance = balance - amount of withdrawal;
   dispense cash;
   store new balance into database;
```

- Ali at ATM1 withdraws $100
- Sara at ATM2 withdraws $50
- Initial balance = $400, final balance = ?
  - Should be $250 no matter who goes first

Sequential Transactions -> Final balance = $250

Ali withdraws $100:

```
read balance; $400
if balance > amount then
    balance = balance - amount; $300
    write balance; $300
```

Sara withdraws $50:

```
read balance; $300
if balance > amount then
    balance = balance - amount; $250
    write balance; $250
```

Concurrent Transactions (Scenario 1) -> Final balance = $300

Ali withdraws $100:

```
read balance; $400
```

```
if balance > amount then
  balance = balance - amount; $300
  write balance; $300
```

Sara withdraws $50:

```
read balance; $400
If balance > amount then
 balance = balance - amount; $350
 write balance; $350
```

Concurrent Transactions (Scenario 2) -> Final balance = $350

Ali withdraws $100:

```
read balance; $400

if balance > amount then
 balance = balance - amount; $300
 write balance; $300
```

Sara withdraws $50:

```
read balance; $400

if balance > amount then
 balance = balance - amount; $350
 write balance; $350
```

Example of consistency issues caused by failures

- **Example to illustrate consistency issues that can be introduced by failures:**

Balance transfer

```
decrement the balance of account X
   by $100;
increment the balance of account Y
   by $100;
```

- Scenario 1: Power goes out after the first instruction
  - **Such failures may leave database in an inconsistent state with partial updates carried out**
  - Transfer of funds from one account to another should either complete or not happen at all

=> Database transactions come to the rescue!

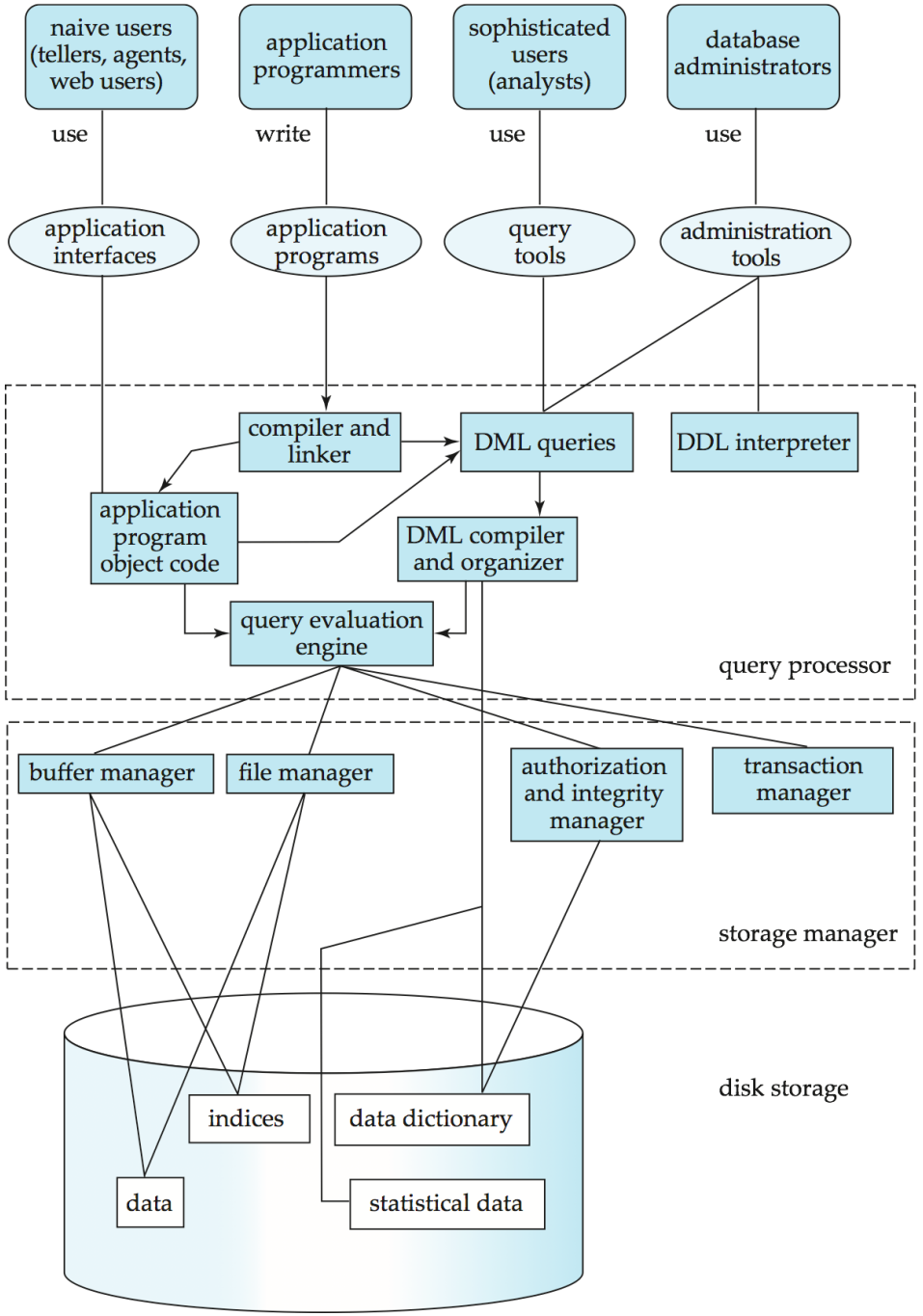
Summary of modern DBMS features

- Persistent storage of data
- Logical data model + declarative queries and updates **=> physical data independence**
  - Provides a declarative interface to data management **=> Hides complexity and increases flexibility**
- Multi-user concurrent access
- Safety from system failures
- Supports high performance:
  - Massive amounts of data (terabytes ~ petabytes)
  - High throughput (thousands ~ millions transactions per minute)
  - High availability (~99.999% uptime)

DBMS Architecture

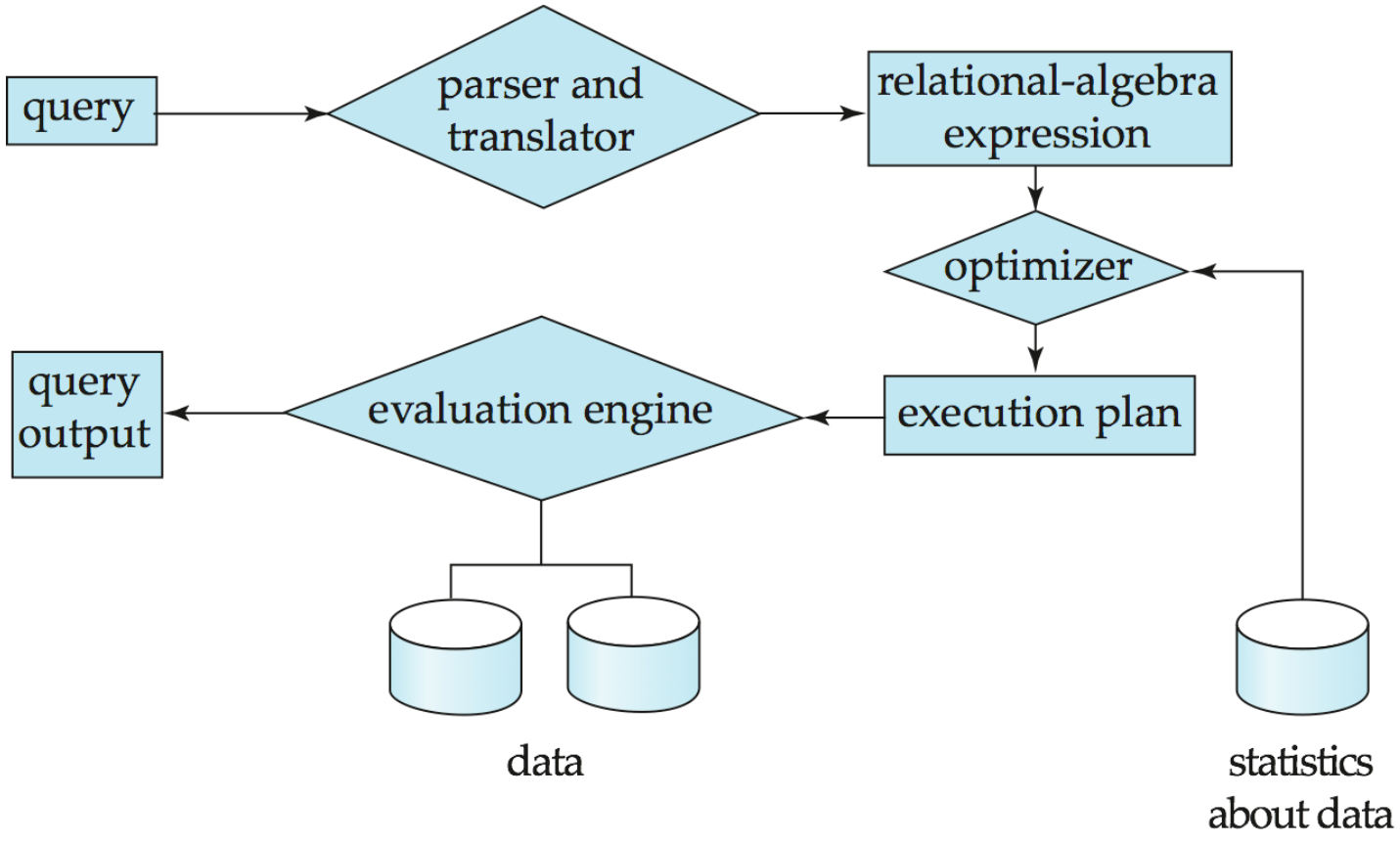
Back

DBMS Architecture



Query Processing

1. Parsing and translation
2. Optimization
3. Evaluation



Query Processing (Cont.)

- Alternative ways of evaluating a given query
  - Equivalent expressions
  - Different algorithms for each operation
- Cost difference between a good and a bad way of evaluating a query can be enormous
- Need to estimate the cost of operations
  - Depends critically on statistical information about relations which the database must maintain
  - Need to estimate statistics for intermediate results to compute cost of complex expressions

Storage Management

- The storage manager is responsible of the following tasks:
  - Interaction with the file manager
  - **Efficient storing, retrieving and updating of data**
- Issues addressed:
  - Storage access
  - File organization
  - Indexing and hashing

Transaction Management

- **Transaction-management component** ensures that the database remains in a consistent (correct) state despite system failures (e.g., power failures and operating system crashes) and transaction failures.
  - A **transaction** is a collection of operations that performs a single logical function in a database application

- **Concurrency-control manager** controls the interaction among the concurrent transactions, to ensure the consistency of the database (when multiple users concurrently update the same data)

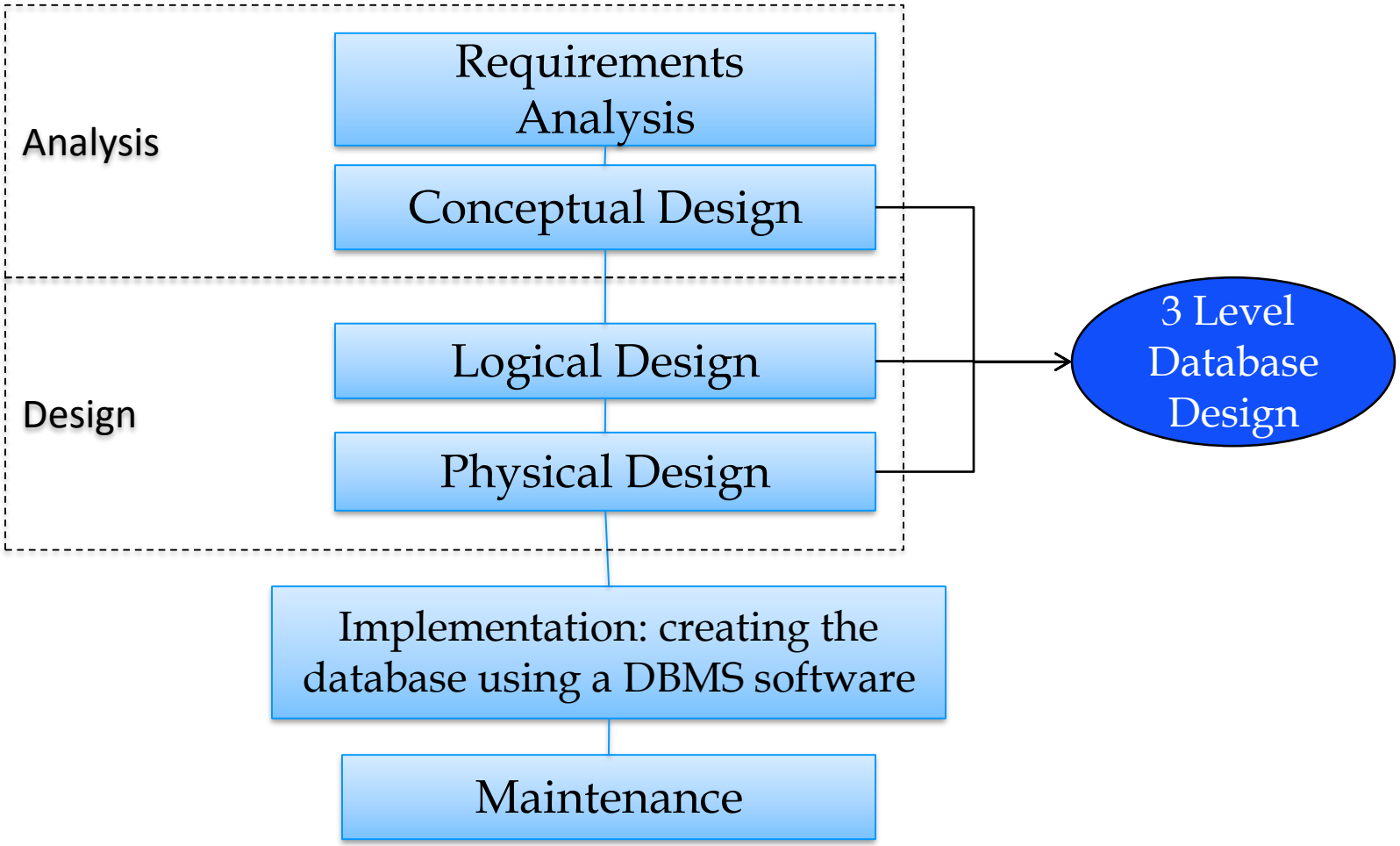
Summary of Relational Database Features

- Storage and access of data that is:
  - **Persistently** stored
  - **Concurrently** accessed
  - **Consistently** modified
  - **Structured** (tabular)
  - **Fast** to access
- Compare: files on disk
  - No concurrency/transactional capabilities (typically)
  - Not as fast (i.e., doesn't scale well)
  - Not structured formally

Conceptual Data Modeling using Entity-Relationship Diagram (ERD)

Back

Database Development Process



→ Similar to software development

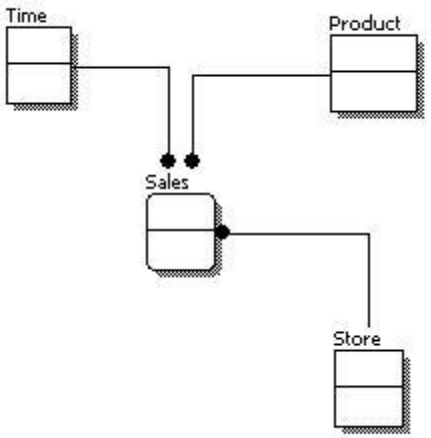
Database Design Steps in Practice

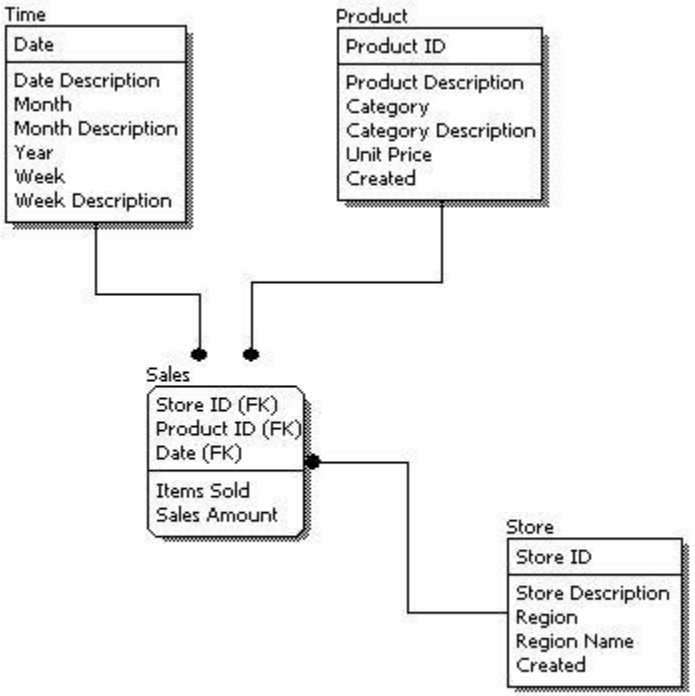
- Requirements Analysis
  - user needs; what must database do?
- Conceptual Design
  - High level design in terms of entities and relationships
- Logical Design
  - Adds further details such as defining what the columns in each table and their datatype.
- Schema Refinement
  - Normalization to reduce redundancy
- Physical Design e.g., indexes to create, add primary keys, foreign keys and constraints to the design
- Security Design - who accesses what

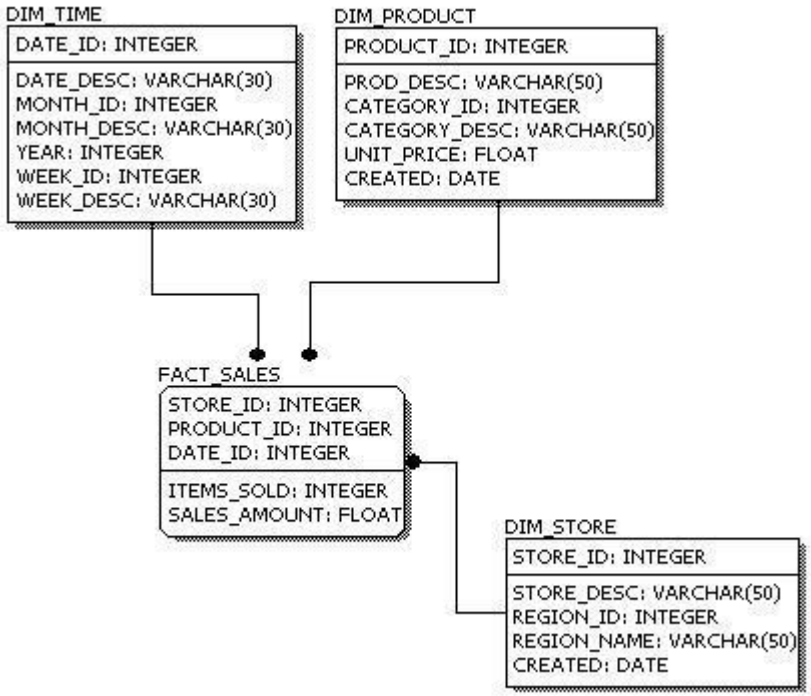
Conceptual, Logical and Physical

Feature	Conceptual	Logical	Physical
Entity Names	✓	✓	
Entity Relationships	✓	✓	
Attributes		✓	
Primary Keys		✓	✓
Foreign Keys		✓	✓
Table Names			✓
Column Names			✓
Column Data Types			✓

Conceptual, Logical and Physical







Conceptual Database Design

- Use the Entity Relationship (ER) model to develop a high level description of the data
- Identify the entities and their relationships

Logical

- Identify what information about these entities and relationships is to be stored in the database
- Identify the integrity constraints that apply to the entities and relationships
- Review the ER model to ensure correctness

Physical Database Design

- Determine which data model should be used to implement the database (relational model vs. NoSQL)
- Determine which DBMS to use
  - In most cases this means deciding which existing DBMS product to purchase or license
- Map, or translate, the logical schema to a database schema of the chosen model
- There are two major problems to be avoided
  - ***Redundancy*** – information should not be repeated
  - ***Incompleteness*** – it should be possible to record all the desired information

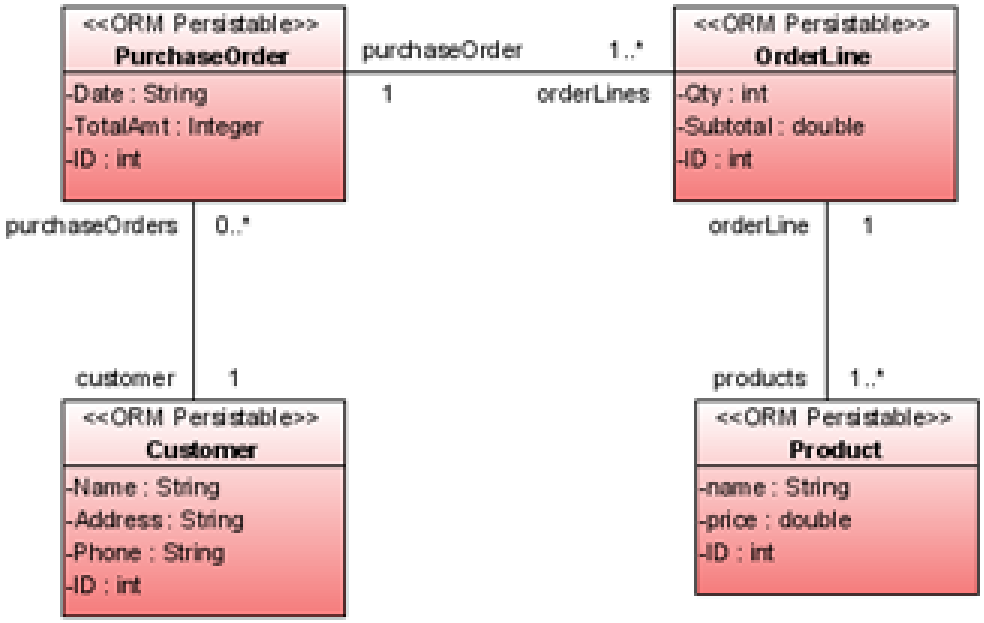
The Entity-Relationship Model

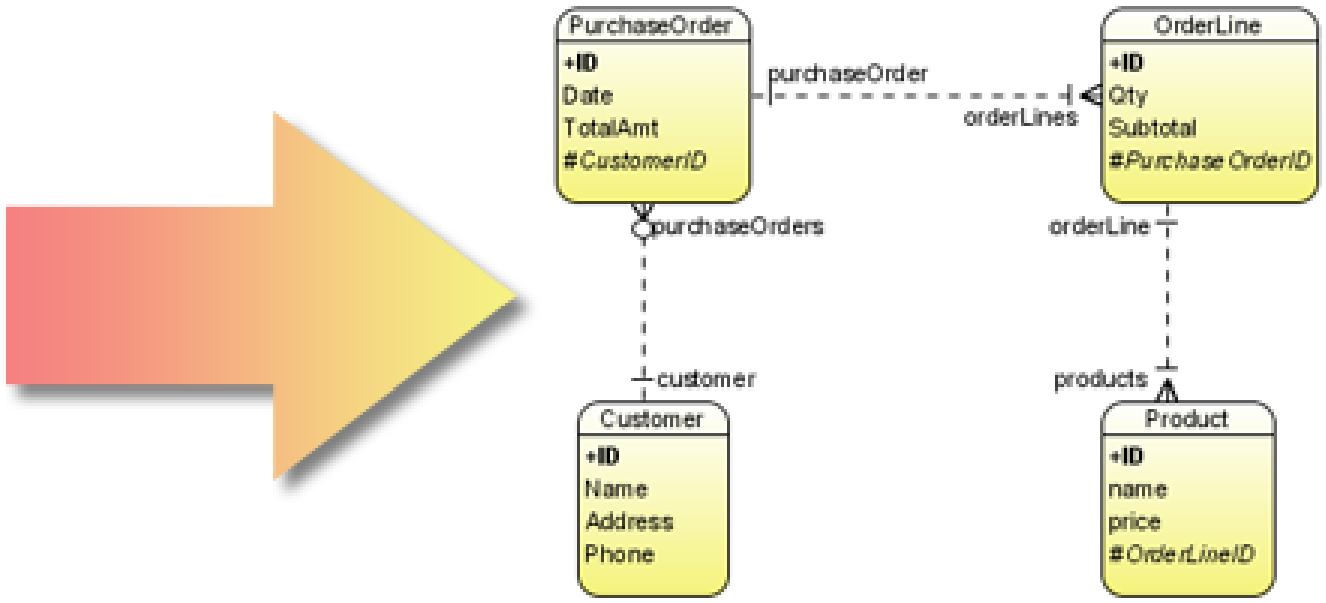
- The most common conceptual data model
- The major components of the ER Model are:
  - **Entity** – something in the real world that we wish to track and store data about (employee, item)
  - **Attribute** – A characteristic of an entity or relationship (EmployeeID, Item Description)
  - **Relationship** – A link that connect between entities (e.g. A customer (*entity*) buys (*relationship*) a product (*entity*)
  - **Constraints** which restrict relationships, e.g. an account *must be* owned by a customer

Three Types of Relationships

- **One-to-many relationships (1:M)**
  - A painter paints many different paintings, but each one of them is painted by only that painter.
    - PAINTER (1) paints PAINTING (M)
- **Many-to-many relationships (M:N)**
  - An employee might learn many job skills, and each job skill might be learned by many employees.
    - EMPLOYEE (M) learns SKILL (N)
- **One-to-one relationships (1:1)**
  - Each store is managed by a single employee and each store manager (employee) only manages a single store.
    - EMPLOYEE (1) manages STORE (1)

Example ER diagram derived from class diagram





SQL Overview

- Standardized language
  - Managed by ANSI
  - But, as usual, the vendors like to innovate
    - Non-standard extensions (data types, syntax structures, etc.)
- Based on relational data model
  - Reading data
    - Match a predicate against a sub-set of the data in tables
    - Results provided as a temporary table of data rows
  - Updating data
    - Match a predicate against a set of data in tables
    - Apply data update operations to every row that matches

SQL Statements

- **C**reating data:

```
INSERT into PERSON (first_name, last_name)
  VALUES ('Ahmed', 'Said')
```

- **R**eading data:

```
SELECT first_name FROM person WHERE last_name =
  'Lee'
```

- **U**pdating data:

```
UPDATE person SET first_name = 'Shrek' where
  last_name = 'Funny'
```

- **D**eleting data:

```
DELETE from person where first_name = 'Simpson'
  and last_name = 'Homer'
```

Review of Normalization

Back

Data Normalization

- The process of decomposing relations with anomalies to produce smaller, well ***structured*** relations free of redundancy:
  - A relation that contains minimal data redundancy allows users to insert, delete, and update rows **without causing data inconsistencies**
- Normalization is a primarily tool to validate and improve a logical design so that it satisfies certain constraints that ***avoid unnecessary duplication of data***
- Goal is to avoid anomalies
  - **Insertion Anomaly** – adding new rows forces user to create duplicate data
  - **Deletion Anomaly** – deleting rows may cause a loss of data that would be needed for other future rows
  - **Modification Anomaly** – changing data in a row forces changes to other rows because of duplication

Student#	Advisor	Adv-Room
1022	Jones	412
4123	Smith	216
4124	Smith	216

Anomalies in this Table

- **Insertion** – can't enter a new student without having the Advisor Room
- **Deletion** – if we remove Student# 1022, we lose information about the room of the Advisor Jones
- **Modification** – Changing the room of the advisor Smith forces us to update multiple records

Why do these anomalies exist?

Because are two entity types were combined. This results in duplication, and an **unnecessary dependency between the entities**

First Normal Form

- No multivalued attributes
  - A table is in first normal form (1NF) if it does not contain a repeating group
- Every attribute value is atomic (singled-value)

Normalized Relations from Employee

EMPLOYEE

Emp_ID	Name	Dept_Name	Salary	Course_Title	Date_Completed
100	Margaret Simpson	Marketing	48,000	SPSS	6/19/2010
100	Margaret Simpson	Marketing	48,000	Surveys	10/7/2010
140	Allen Beeton	Accounting	52,000	Tax Acc	12/8/2010
110	Chris Lucero	Info. System	43,000	SPSS	1/12/2010
110	Chris Lucero	Info. System	43,000	C++	4/22/2010
190	Lorenzo Davis	Finance	55,000		
150	Susan Martin	Marketing	42,000	SPSS	6/16/2010
150	Susan Martin	Marketing	42,000	Java	8/12/2010

EMPLOYEE

Emp_ID	Name	Dept_Name	Salary
100	Margaret Simpson	Marketing	48,000
140	Allen Beet	Accounting	52,000
110	Chris Lucero	Info. System	43,000
190	Lorenzo Davis	Finance	55,000
150	Sususan Martin	Marketing	42,000

Is there any ano-maly?

EMP_COURSE

Emp_ID	Course_Title	Date_Completed
100	SPSS	6/19/2010
100	Surveys	10/7/2010
140	Tax Acc	12/8/2010
110	SPSS	1/12/2010
110	C++	4/22/2010
150	SPSS	6/19/2010
150	Java	8/12/2010

Second Normal Form

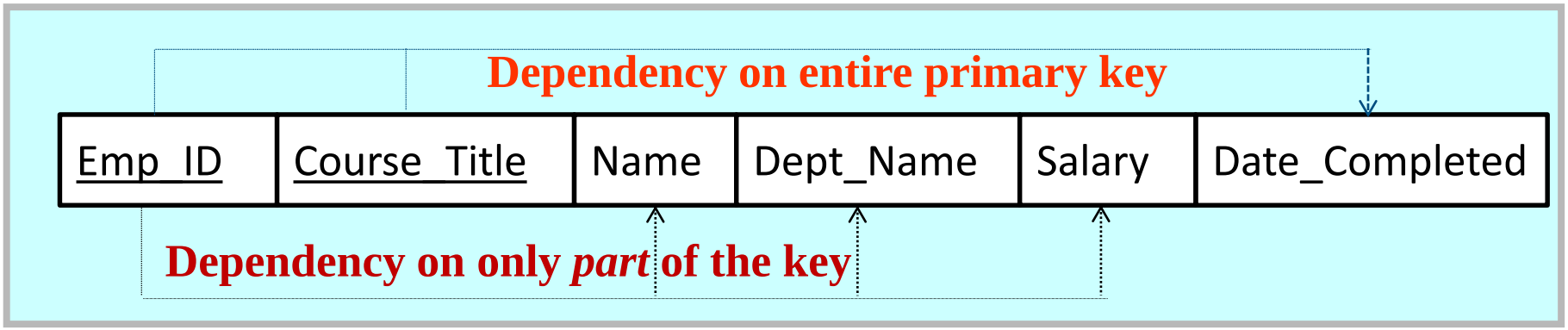
- 1NF and **every** non-key attribute is functionally dependent on the entire primary key.
- No partial functional dependencies: Every non-key attribute must be defined by the entire key not by only part of the key.

EXAMPLE OF FUNCTIONAL DEPENDENCY

EMPLOYEE NUMBER	PROJECT CODE	EMPLOYEE NAME	PROJECT NAME	TOTAL DAYS ON PROJECT

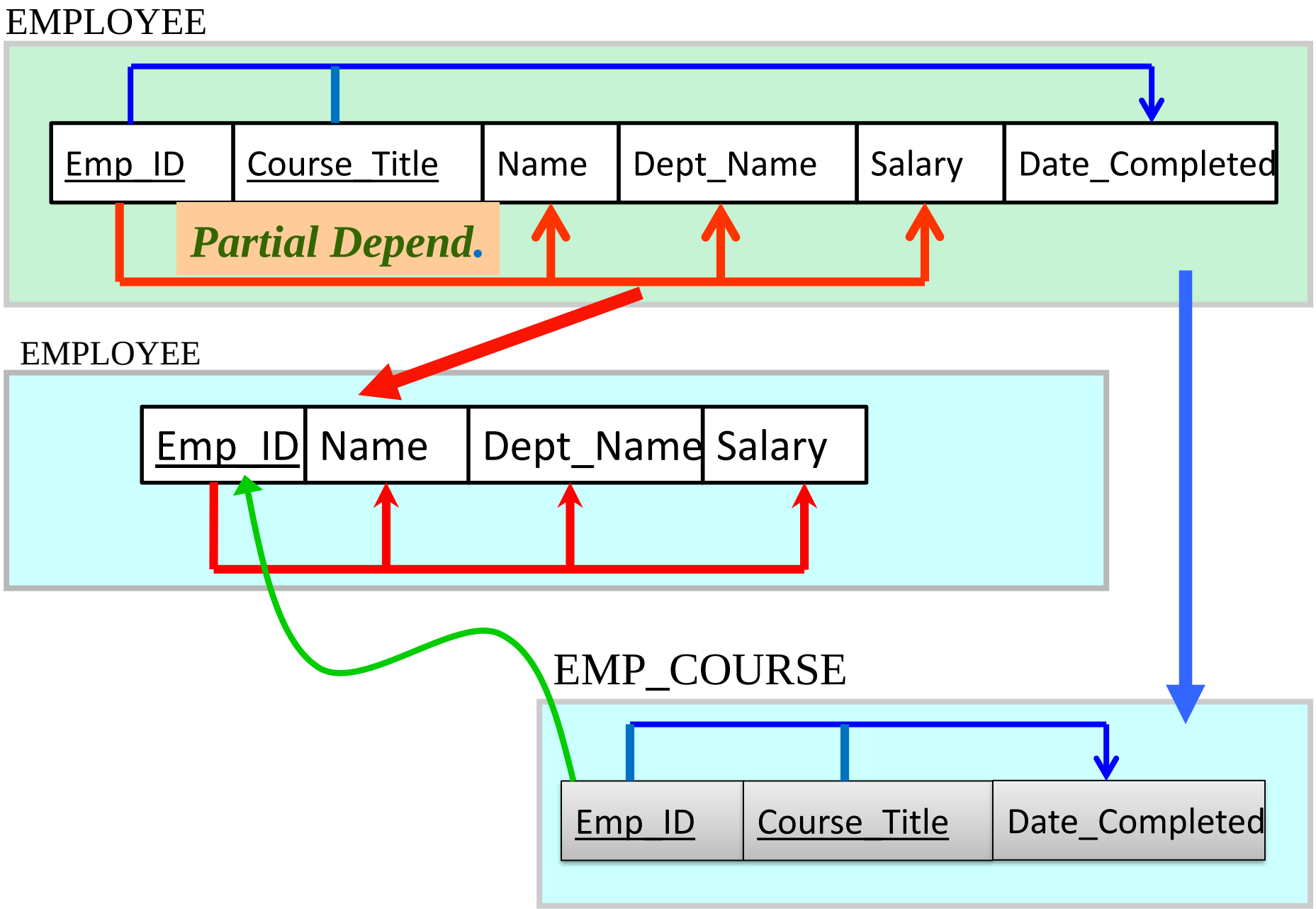
EMPLOYEE NAME	is functionally dependent on EMPLOYEE NUMBER but not on PROJECT CODE
PROJECT NAME	is functionally dependent on PROJECT CODE but not on EMPLOYEE NUMBER
TOTAL DAYS ON PROJECT	is functionally dependent on the concatenated key of EMPLOYEE NUMBER and PROJECT CODE

Functional Dependencies in EMPLOYEE



Therefore, NOT in 2nd Normal Form!!

Normalization from 1NF to 2NF

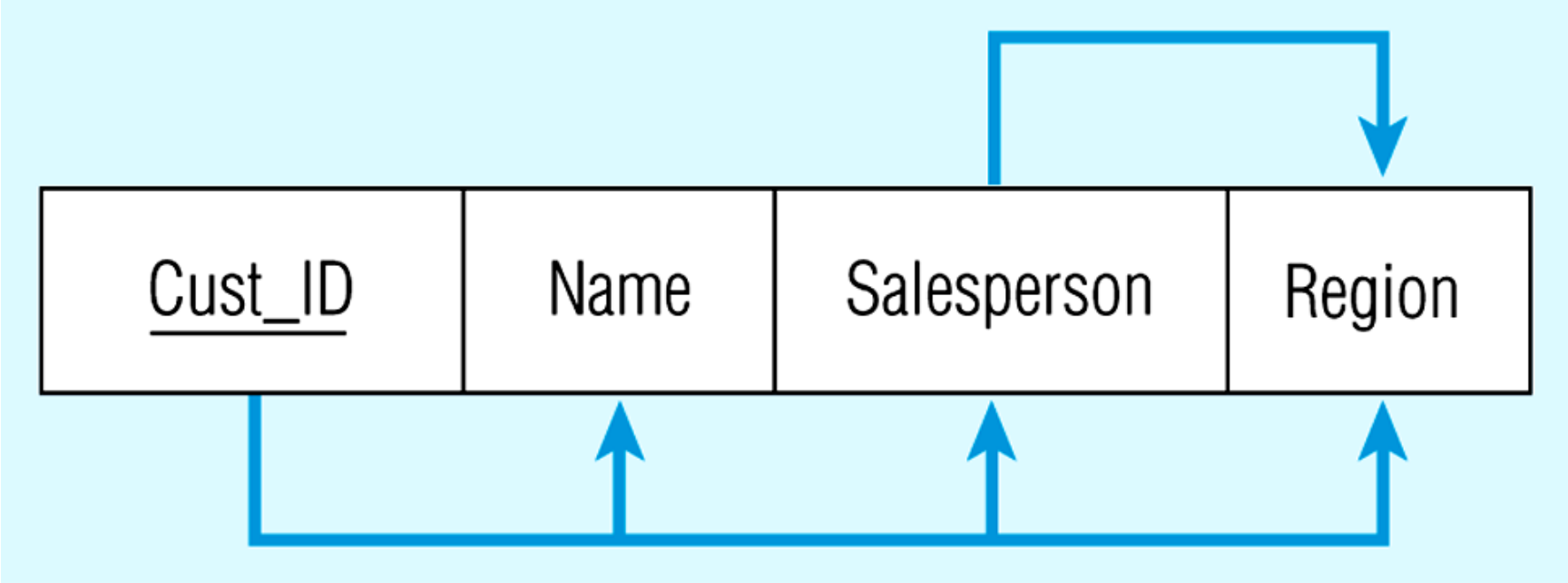


Third Normal Form

- 2NF and no transitive dependencies (no functional dependency between non-key attributes)

=> all fields are functionally dependent ONLY on the primary key

Relation with transitive dependency

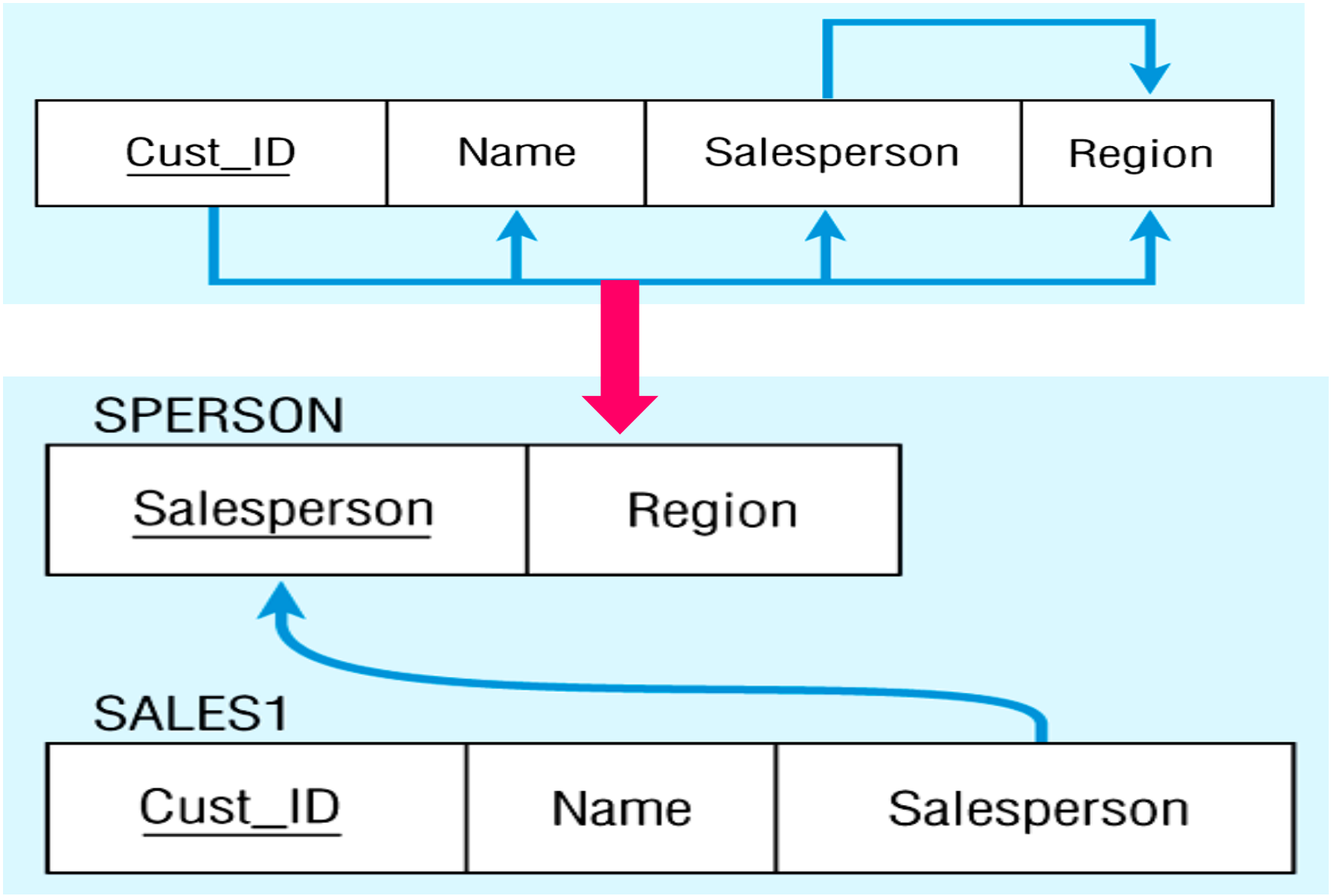


CustID → Name
CustID → Salesperson
CustID → Region
and
Salesperson → Region
All this is OK
(2nd NF)

BUT
CustID → Salesperson → Region
implies
CustID → Region
Transitive dependency
(not in 3rd NF)

Relations in 3NF

Remove a transitive dependency



Removing a transitive dependency

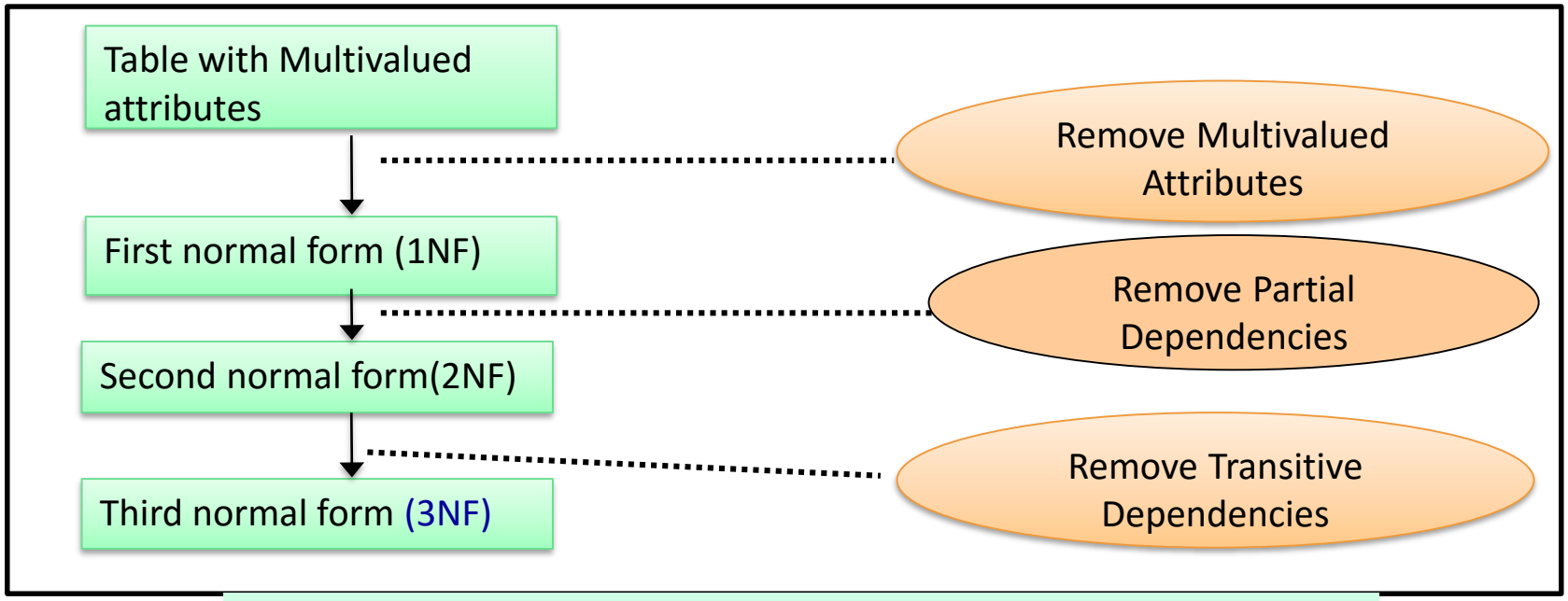
(a) Decomposing the SALES relation

SALES1

Cust_ID	**Name**	**Salesperson**
8023	Anderson	101
9167	Bancroft	102
7924	Hobbs	101
6837	Tucker	103
8596	Eckersley	102
7018	Arnold	104

S_PERSON

Salesperson	**Region**
101	South
102	West
103	East
104	North



The Normalisation Process

Normalisation

1. Check for multi-valued attributes
   If you find any, restructure table to remove them
   table is now in 1st NF
2. Check for partial dependencies
   If you find any, restructure table to remove them
   table is now in 2nd NF
3. Check for transitive dependencies
   If you find any, restructure table to remove them
   table is now in 3rd NF