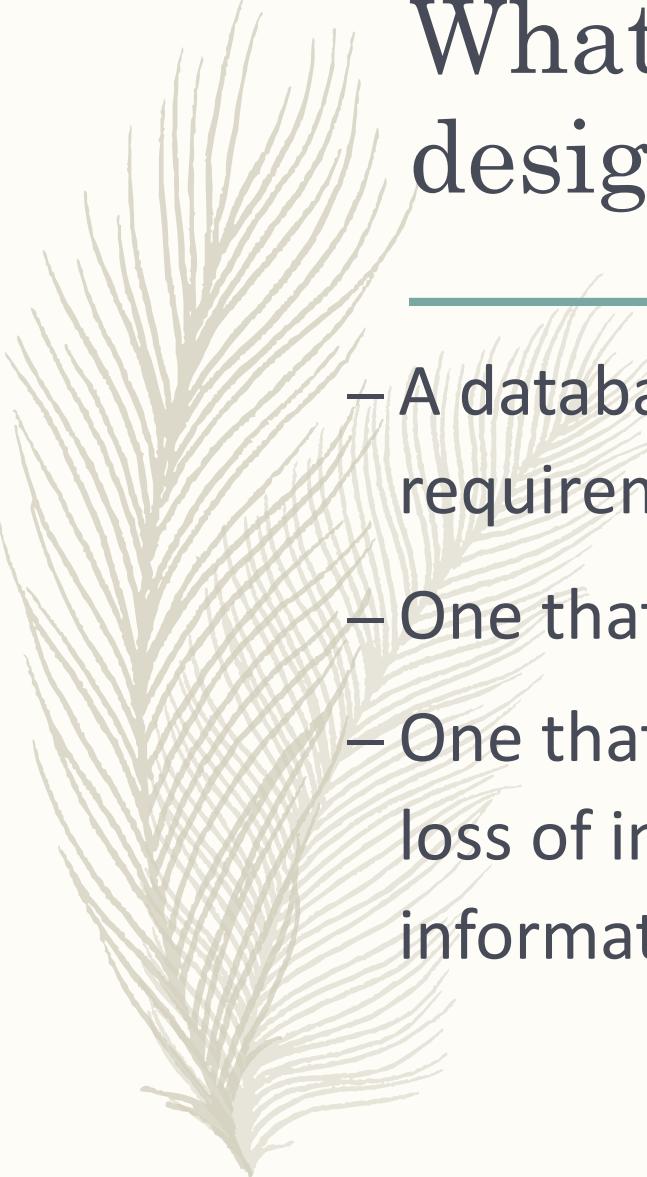




# Normalization

---

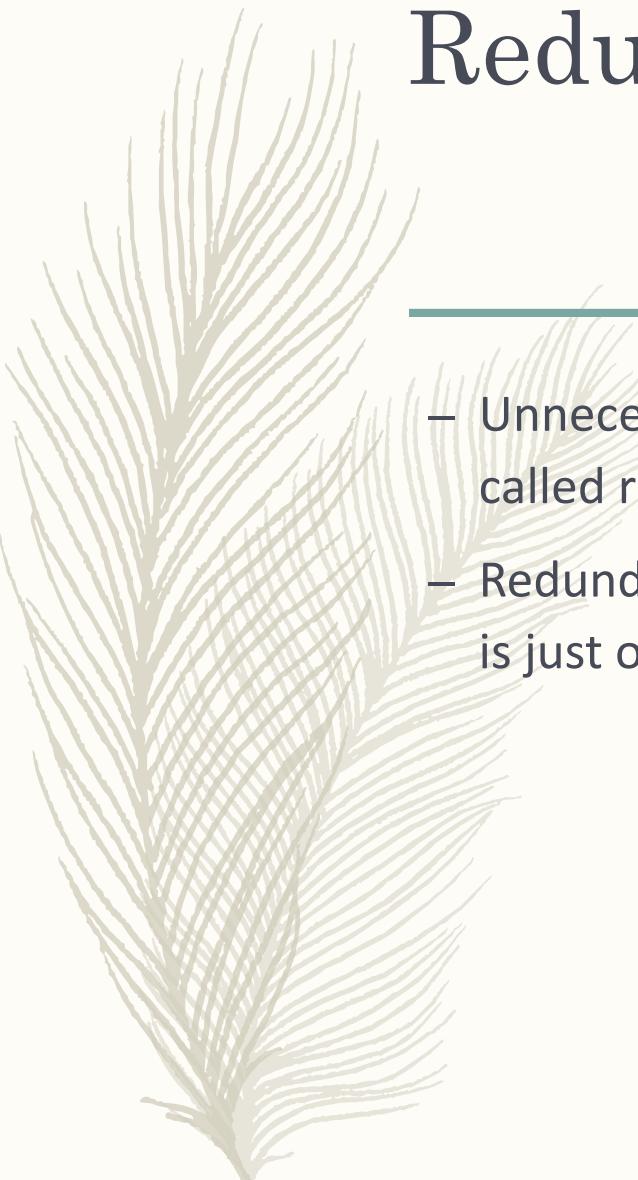
Pritesh Patel



# What is a good database design?

---

- A database design in which all the user requirements are satisfied
- One that does not waste storage space
- One that does not lead to misinformation, loss of information or inconsistent information



# Redundancy

---

- Unnecessary repetition of information in the database is called redundancy.
- Redundancy results in several problems, wastage of space is just one of them.

# Problems Associated with Redundancy

---

- Update anomalies
- Insertion anomalies
- Modification anomalies
- Deletion anomalies

# Sample Table (DeptEmp)

| <u>Emp#</u> | <u>Ename</u> | <u>Salary</u> | <u>Dept#</u> | <u>Dname</u> | <u>Dloc</u> |
|-------------|--------------|---------------|--------------|--------------|-------------|
| 101         | aaa          | 10000         | 2            | Acc          | HQ          |
| 102         | bbb          | 7000          | 1            | Stores       | Plant       |
| 103         | ccc          | 12000         | 2            | Acc          | HQ          |
| 104         | ddd          | 9000          | 3            | HR           | HQ          |

# The Database After Normalization

Emp

| <u>Emp#</u> | <u>Ename</u> | <u>Salary</u> | <u>Dept#</u> |
|-------------|--------------|---------------|--------------|
| 101         | aaa          | 10000         | 2            |
| 102         | bbb          | 7000          | 1            |
| 103         | ccc          | 12000         | 2            |
| 104         | ddd          | 9000          | 3            |

Dept

| <u>Dept#</u> | <u>Dname</u> | <u>Dloc</u> |
|--------------|--------------|-------------|
| 1            | Stores       | Plant       |
| 2            | Acc          | HQ          |
| 3            | HR           | HQ          |

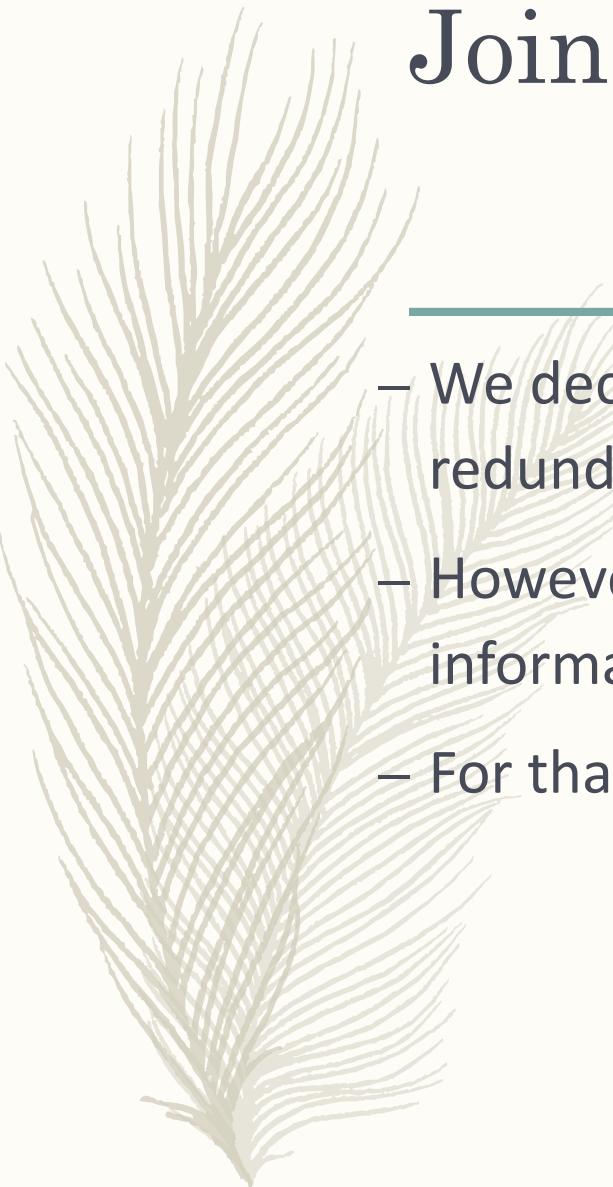
# Wrong Database Design

Emp

| <u>Emp#</u> | <u>Ename</u> | <u>Salary</u> |
|-------------|--------------|---------------|
| 101         | aaa          | 10000         |
| 102         | bbb          | 7000          |
| 103         | ccc          | 12000         |
| 104         | ddd          | 9000          |

Dept

| <u>Dept#</u> | <u>Dname</u> | <u>Dloc</u> |
|--------------|--------------|-------------|
| 1            | Stores       | Plant       |
| 2            | Acc          | HQ          |
| 3            | HR           | HQ          |



# Join

---

- We decompose a table into multiple tables to avoid redundancy and associated problems.
- However, Sometimes we need to put together information from multiple tables.
- For that, we perform a join operation.



# The Basis for Performing a Join

---

- Join is always performed based on some column(s) common between the two tables.
- These column(s) always form primary key – foreign key relationship between the two tables, i.e. they are primary/unique key in one table and foreign key in the other.

# The Example Database

Emp

| <u>Emp#</u> | <u>Ename</u> | <u>Salary</u> | <u>Dept#</u> |
|-------------|--------------|---------------|--------------|
| 101         | aaa          | 10000         | 2            |
| 102         | bbb          | 7000          | 1            |
| 103         | ccc          | 12000         | 2            |
| 104         | ddd          | 9000          | 3            |

Dept

| <u>Dept#</u> | <u>Dname</u> | <u>Dloc</u> |
|--------------|--------------|-------------|
| 1            | Stores       | Plant       |
| 2            | Acc          | HQ          |
| 3            | HR           | HQ          |



# The Link Between the Tables

---

- The column “Dept#” is common between the tables.
- It is the primary key in the Dept table and foreign key in the Emp table.
- It is the link between the two tables.

# How Will the Two Tables be Joined?

---

- The rows from the Emp table will be taken one-by-one and the Dept# column in it will be examined.
- The row containing the same value in the Dept# column of the Dept table will be joined with the corresponding row in the Emp table.

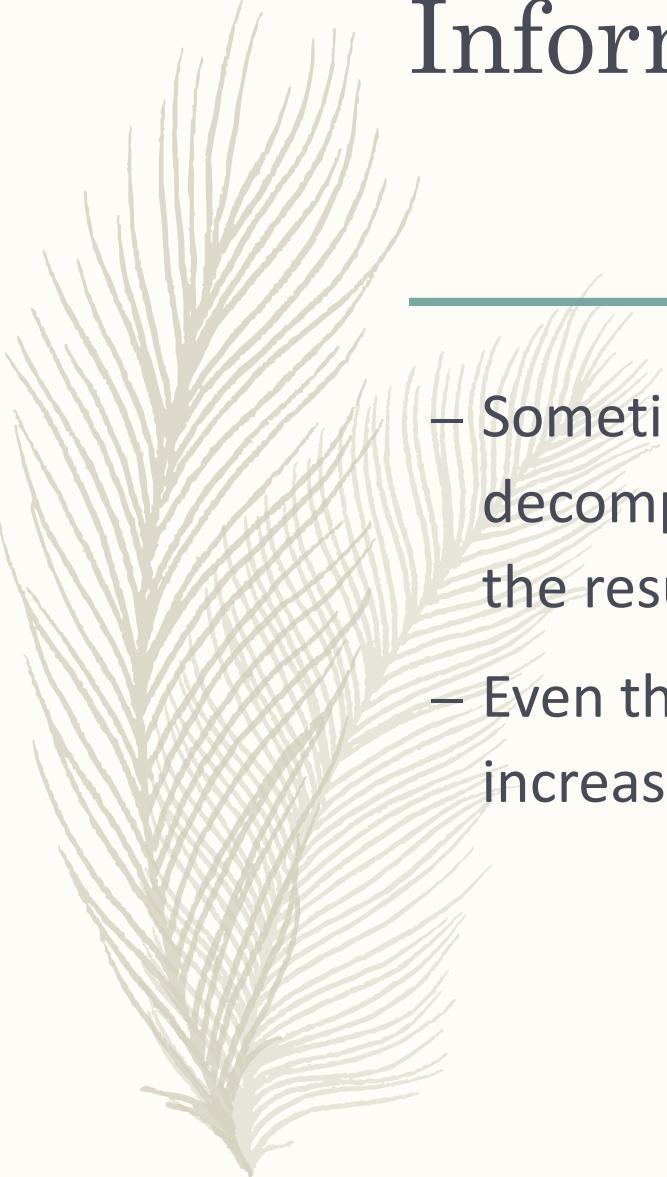
# The Result of Joining the Two Tables

Emp

| <u>Emp#</u> | <u>Ename</u> | <u>Salary</u> | <u>Dept#</u> | <u>Dept#</u> | <u>Dname</u> | <u>Dloc</u> |
|-------------|--------------|---------------|--------------|--------------|--------------|-------------|
| 101         | aaa          | 10000         | 2            | 2            | Acc          | HQ          |
| 102         | bbb          | 7000          | 1            | 1            | Stores       | Plant       |
| 103         | ccc          | 12000         | 2            | 2            | Acc          | HQ          |
| 104         | ddd          | 9000          | 3            | 3            | HR           | HQ          |

Dept

| <u>Dept#</u> | <u>Dname</u> | <u>Dloc</u> |
|--------------|--------------|-------------|
| 1            | Stores       | Plant       |
| 2            | Acc          | HQ          |
| 3            | HR           | HQ          |



# Information Loss

---

- Sometimes, if the tables are improperly decomposed, spurious rows are generated in the result of the join.
- Even though the gross quantity of data increases, there is actually information loss.

# The Example of Information Loss

Emp

| <u>Emp#</u> | <u>Ename</u> | <u>Salary</u> | <u>Dloc</u> |
|-------------|--------------|---------------|-------------|
| 101         | aaa          | 10000         | HQ          |
| 102         | bbb          | 7000          | Plant       |
| 103         | ccc          | 12000         | HQ          |
| 104         | ddd          | 9000          | HQ          |

Dept

| <u>Dept#</u> | <u>Dname</u> | <u>Dloc</u> |
|--------------|--------------|-------------|
| 1            | Stores       | Plant       |
| 2            | Acc          | HQ          |
| 3            | HR           | HQ          |

# Result of Joining

Joined Table

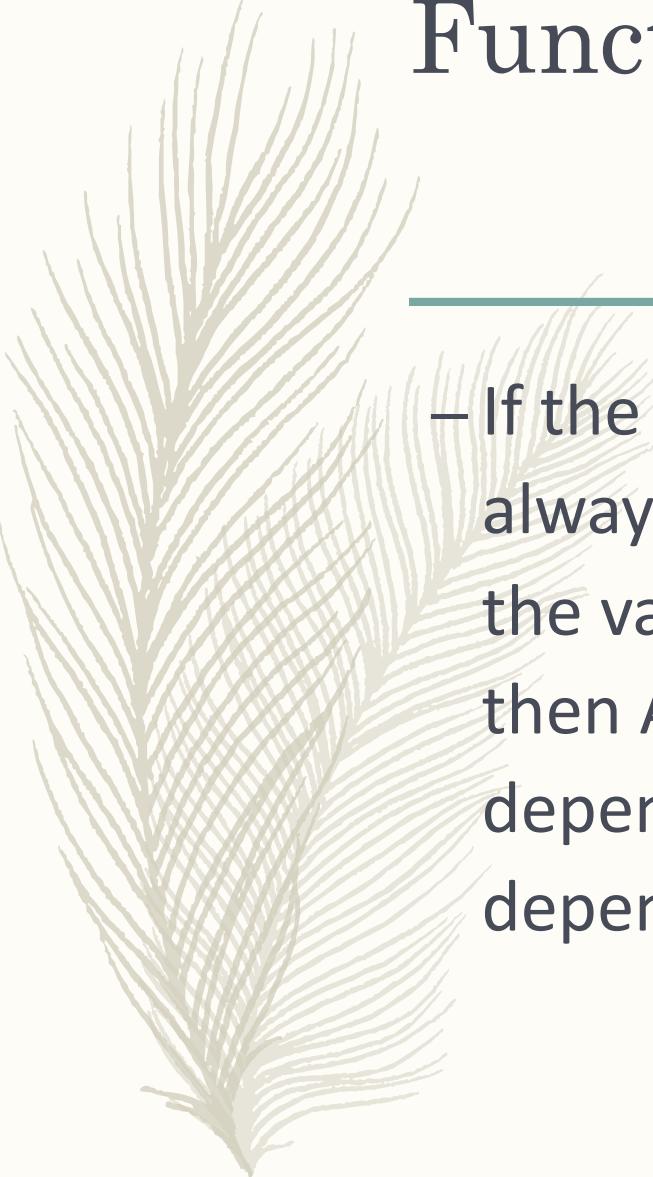
| Emp# | Ename | Salary | Dloc  | Dept# | Dname  | Dloc  |
|------|-------|--------|-------|-------|--------|-------|
| 101  | aaa   | 10000  | HQ    | 2     | Acc    | HQ    |
| 101  | aaa   | 10000  | HQ    | 3     | HR     | HQ    |
| 102  | bbb   | 7000   | Plant | 1     | Stores | Plant |
| 103  | ccc   | 12000  | HQ    | 2     | Acc    | HQ    |
| 103  | ccc   | 12000  | HQ    | 3     | HR     | HQ    |
| 104  | ddd   | 9000   | HQ    | 2     | Acc    | HQ    |
| 104  | ddd   | 9000   | HQ    | 3     | HR     | HQ    |



# Lossless Join

---

- Lossless join means a table, when decomposition is necessary, must be decomposed in such a way that if we join the decomposed table back, we always get the original table.
- Lossless join is an important requirement in database design.



# Functional Dependencies

---

- If the values in an attribute-set A can always be determined (derived) from the values in another attribute-set X, then A is said to be functionally dependent on X and the functional dependency is written as  $X \rightarrow A$ .

# Examples of Functional Dependencies

---

- BookMaster (Book#, Title, Authors, Edition, Publisher, Category, ISBN#)
- Contacts (ContactId, Name, Sex, BirthDate, Category, Relation, Addr1, Addr2, Addr3, City, PIN, State, Country, STDCode, Phones, Mobile, SpouseName, SpouseBirthDate)



# Types of Normalization

---

- 1NF (1<sup>st</sup> Normal Form)
- 2NF (2<sup>nd</sup> Normal Form)
- 3NF (3<sup>rd</sup> Normal Form)
- BCNF (Boyce-Codd Normal Form)
- 4NF (4<sup>th</sup> Normal Form)
- 5NF (5<sup>th</sup> Normal Form) aka PJNF (Project-Join Normal Form)
- DKNF (Domain-Key Normal Form)

# The First Normal Form (1NF)

---

- All values in the cells should be atomic (i.e. indivisible).
- Multivalued attributes are not allowed.
- Composite attributes are not allowed.
- No “tables within tables” (nested relations).

# The Supplier Example

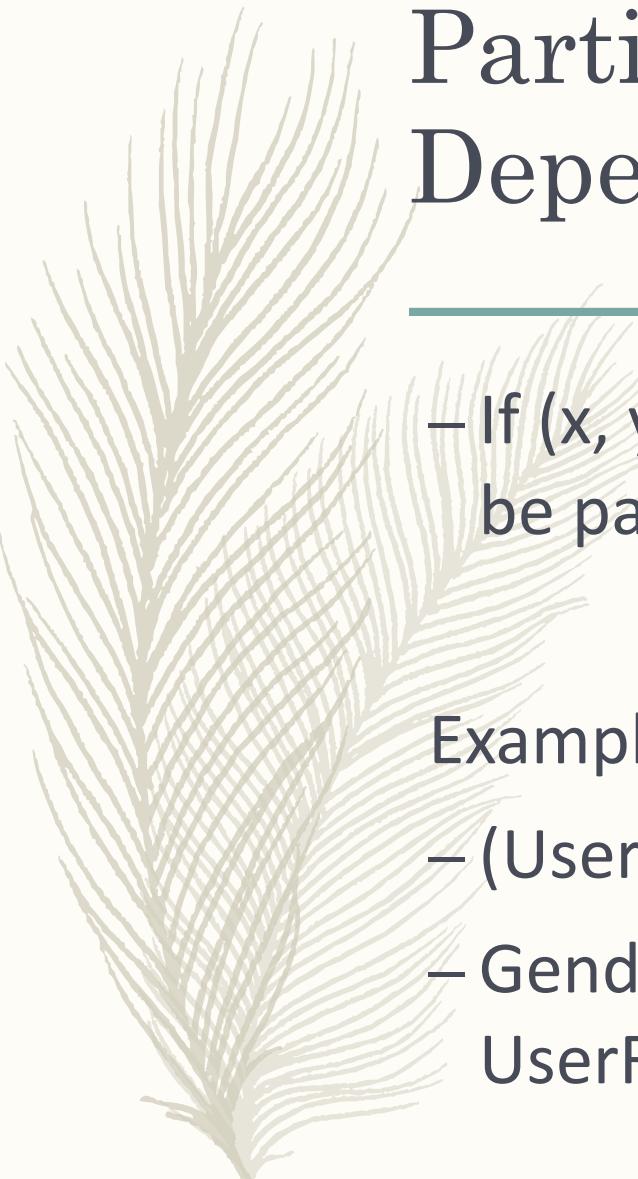
| <u>Sup#</u> | <u>SName</u> | <u>Items</u> |              |              |
|-------------|--------------|--------------|--------------|--------------|
|             |              | <u>Item#</u> | <u>IName</u> | <u>Price</u> |
| 1           | A Ltd.       | 101          | aaa          | 100          |
|             |              | 102          | bbb          | 200          |
|             |              | 103          | ccc          | 300          |
| 2           | B Ltd.       | 104          | ddd          | 150          |
|             |              | 101          | aaa          | 100          |

# The Supplier Example

| <u>Sup#</u> | <u>SName</u> |
|-------------|--------------|
| 1           | A Ltd.       |
| 2           | B Ltd.       |

| <u>Item#</u> | <u>IName</u> | <u>Price</u> |
|--------------|--------------|--------------|
| 101          | aaa          | 100          |
| 102          | bbb          | 200          |
| 103          | ccc          | 300          |
| 104          | ddd          | 150          |

| <u>Sup#</u> | <u>Item#</u> |
|-------------|--------------|
| 1           | 101          |
| 1           | 102          |
| 1           | 103          |
| 2           | 104          |
| 2           | 101          |



# Partial Functional Dependency

---

- If  $(x, y) \rightarrow z$  and also  $y \rightarrow z$  then  $z$  is said to be partially dependent on  $(x, y)$ .

Example:

- (`UserRegNo`, `PhoneNo`, `Gender`)
- `Gender` is partially dependent on `UserRegNo` (not on `PhoneNo`).



# Full Functional Dependency

---

- If  $(x, y) \rightarrow z$  and neither  $x \rightarrow z$  nor  $y \rightarrow z$  then  $z$  is said to be fully dependent on  $(x, y)$ .

Example:

- $(\text{EmplID}, \text{ProjectID}, \text{Hours})$
- Hours is fully FD on EmplID and ProjectID.



# The Second Normal Form (2NF)

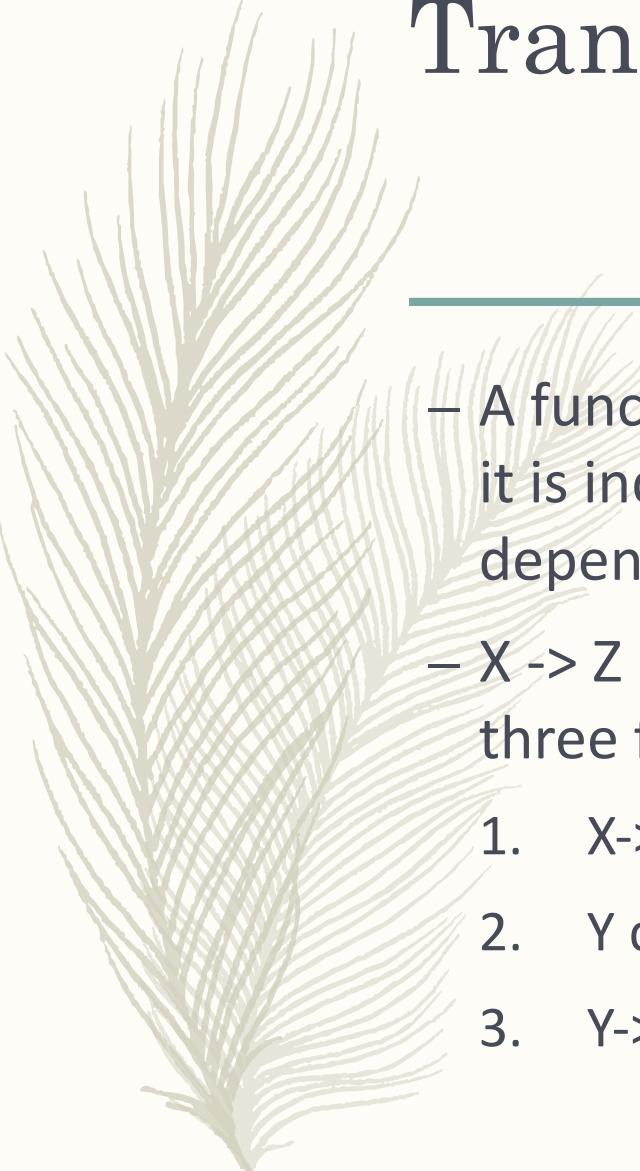
---

- No non-key attribute should be dependent on a part of a key
- All non-key attributes should be fully functionally dependent on the key(s)
- Partial dependencies are not allowed

# Example of 2NF Violation & Solution

---

- SupplierAccounts (Sup#, Sname, Item#, Iname, Price)
- The Solution:
  - Supplier (Sup#, Sname)
  - Item (Item#, Iname, Price)
  - SupItem (Sup#, Item#)



# Transitive Dependency

---

- A functional dependency is said to be transitive if it is indirectly formed by two functional dependencies.
- $X \rightarrow Z$  is a transitive dependency if the following three functional dependencies hold true:
  1.  $X \rightarrow Y$
  2.  $Y$  does not  $\rightarrow X$
  3.  $Y \rightarrow Z$



# Example of Transitive Dependency

---

## Example:

- {Book} ->{Author} (if we know the book, we know the author name)
- {Author} does not ->{Book}
- {Author} -> {Author\_nationality}

# The Third Normal Form (3NF)...

---

- 2NF should be satisfied.
- No non-key attribute should be dependent on another non-key attribute.
- No non-key attribute should be transitively dependent on the key.

# ...The Third Normal Form (3NF)

---

- All non-key attributes should be fully functionally dependent on the key and key only.
- If  $X \rightarrow A$  is a functional dependency, then either X should be a super key or A should be a key attribute.

# Example of 3NF Violation & Solution

---

- AccountMaster (AcType, Ac#, Cust#, CustName, Balance)
- The Solution:
  - AccountMaster (AcType, Ac#, Cust#, Balance)
  - CustomerMaster (Cust#, CustName)

# The Boyce-Codd Normal Form (BCNF)...

---

- 3NF should be satisfied
- No (key or non-key) attribute should be dependent on another non-key attribute.
- No (key or non-key) attribute should be transitively dependent on the key

# ...The Boyce-Codd Normal Form (BCNF)

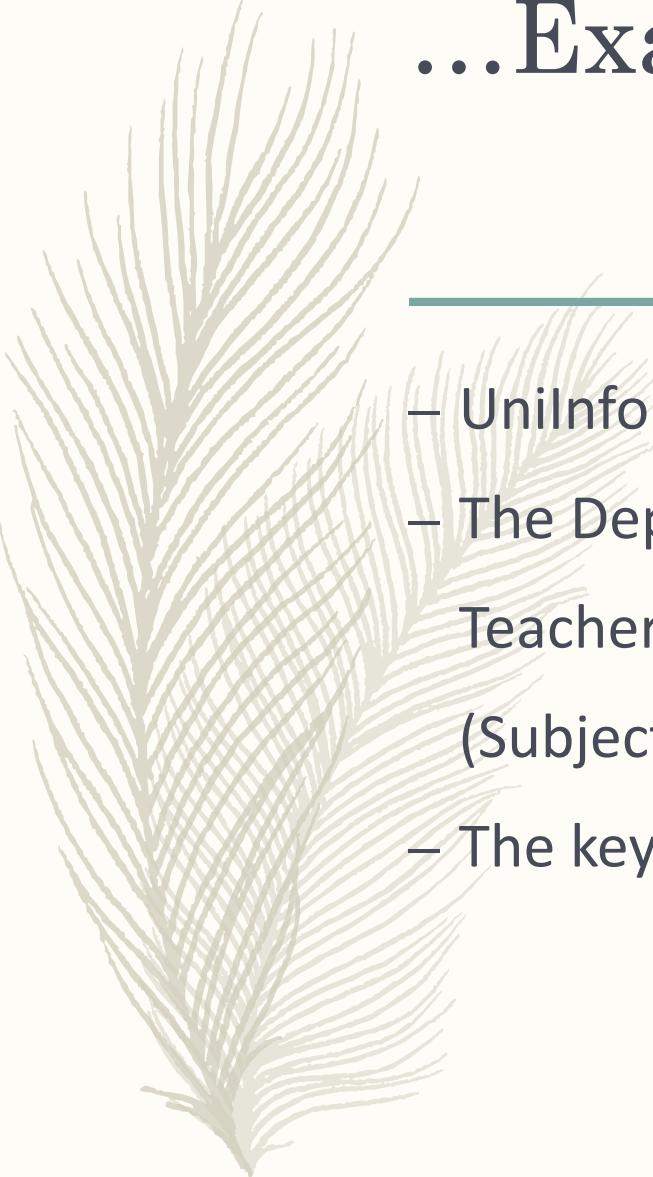
---

- All (key and non-key) attributes should be fully functionally dependent on the key and key only.
- If  $X \rightarrow A$  is a functional dependency, then X must be a super key

# Example of BCNF...

---

- Consider a University.
- One teacher can teach one subject only.
- A subject may be taught by one or more teachers.
- A student can take one or more subjects.



# ...Example of BCNF...

---

- UnilInfo (Teacher, Subject, Student)
- The Dependencies:
  - Teacher → Subject
  - (Subject, Student) → Teacher
- The key: (Subject, Student)



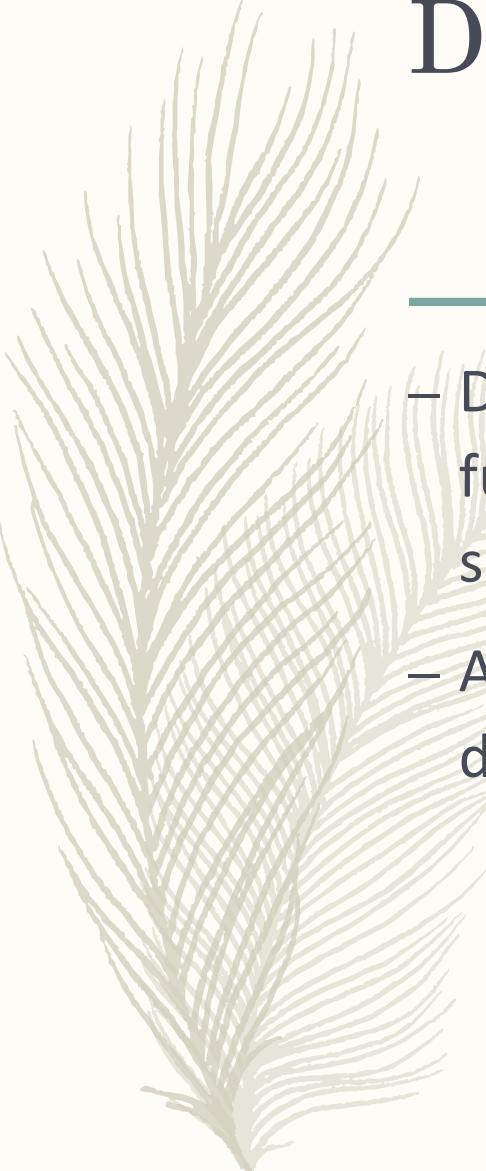
# ...Example of BCNF...

---

- Satisfies 3NF
- Does not satisfy BCNF
- The Solution:

TeacherSubject (Teacher, Subject)

TeacherStudent (Teacher, Student)



# Dependency Preservation

---

- Dependency preservation means each functional dependency in a database design should be testable using one table only.
- All the attributes involved in a functional dependency should be part of the same table.

# BCNF and Dependency Preservation

---

- It is not always possible to produce a design that satisfies dependency preservation as well as BCNF.
- Our previous solution that was in BCNF does not satisfy dependency preservation.

# 3NF and Dependency Preservation

---

- It can be proved that it is always possible to produce a design that satisfies dependency preservation as well as 3NF.
- Our previous example, before decomposition, satisfied dependency preservation.

# Important Goals in Normalization

---

- To achieve the highest normal form
- Lossless join (Compulsory)
- Dependency preservation (optional)

# Example Normalization

## Sales Records:

| Cust Name   | Item              | Shipping Address     | Newsletter                  | Supplier  | Supplier Phone | Price |
|-------------|-------------------|----------------------|-----------------------------|-----------|----------------|-------|
| Alan Smith  | Xbox One          | 35 Palm St, Miami    | Xbox News                   | Microsoft | (800) BUY-XBOX | 250   |
| Roger Banks | PlayStation 4     | 47 Campus Rd, Boston | PlayStation News            | Sony      | (800) BUY-SONY | 300   |
| Evan Wilson | Xbox One, PS Vita | 28 Rock Av, Denver   | Xbox News, PlayStation News | Wholesale | Toll Free      | 450   |
| Alan Smith  | PlayStation 4     | 47 Campus Rd, Boston | PlayStation News            | Sony      | (800) BUY-SONY | 300   |

# 1<sup>st</sup> Normal Form

## Sales Records:

| Cust Name   | Item              | Shipping Address     | Newsletter                  | Supplier  | Supplier Phone | Price |
|-------------|-------------------|----------------------|-----------------------------|-----------|----------------|-------|
| Alan Smith  | Xbox One          | 35 Palm St, Miami    | Xbox News                   | Microsoft | (800) BUY-XBOX | 250   |
| Roger Banks | PlayStation 4     | 47 Campus Rd, Boston | PlayStation News            | Sony      | (800) BUY-SONY | 300   |
| Evan Wilson | Xbox One, PS Vita | 28 Rock Av, Denver   | Xbox News, PlayStation News | Wholesale | Toll Free      | 450   |
| Alan Smith  | PlayStation 4     | 47 Campus Rd, Boston | PlayStation News            | Sony      | (800) BUY-SONY | 300   |

- 1<sup>st</sup> Normal Form**
- Each cell to be Single valued
  - Entries in a column are same type
  - Rows uniquely identified - Add Unique ID, or Add more columns to make unique  
(Note: The order of the rows and the order of the columns are irrelevant)

Primary Key

| Cust ID  | Cust Name   | Item          | Shipping Address     | Newsletter       | Supplier  | Supplier Phone | Price |
|----------|-------------|---------------|----------------------|------------------|-----------|----------------|-------|
| at_smith | Alan Smith  | Xbox One      | 35 Palm St, Miami    | Xbox News        | Microsoft | (800) BUY-XBOX | 250   |
| roger25  | Roger Banks | PlayStation 4 | 47 Campus Rd, Boston | PlayStation News | Sony      | (800) BUY-SONY | 300   |
| wilson44 | Evan Wilson | Xbox One      | 28 Rock Av, Denver   | Xbox News        | Microsoft | (800) BUY-XBOX | 250   |
| wilson44 | Evan Wilson | PS Vita       | 28 Rock Av, Denver   | PlayStation News | Sony      | (800) BUY-SONY | 200   |
| am_smith | Alan Smith  | PlayStation 4 | 47 Campus Rd, Boston | PlayStation News | Sony      | (800) BUY-SONY | 300   |

# 2<sup>nd</sup> Normal Form

2<sup>nd</sup> Normal Form - All attributes (Non-Key Columns) dependent on the key

Primary Key

| Cust ID  | Cust Name   | Shipping Address     | Newsletter       |
|----------|-------------|----------------------|------------------|
| at_smith | Alan Smith  | 35 Palm St, Miami    | Xbox News        |
| roger25  | Roger Banks | 47 Campus Rd, Boston | PlayStation News |
| wilson44 | Evan Wilson | 28 Rock Av, Denver   | Xbox News        |
| wilson44 | Evan Wilson | 28 Rock Av, Denver   | PlayStation News |
| am_smith | Alan Smith  | 47 Campus Rd, Boston | PlayStation News |

Primary Key

| Item          | Supplier  | Supplier Phone | Price |
|---------------|-----------|----------------|-------|
| Xbox One      | Microsoft | (800) BUY-XBOX | 250   |
| PlayStation 4 | Sony      | (800) BUY-SONY | 300   |
| PS Vita       | Sony      | (800) BUY-SONY | 200   |

Primary Key

| Cust ID  | Item          |
|----------|---------------|
| at_smith | Xbox One      |
| roger25  | PlayStation 4 |
| wilson44 | Xbox One      |
| wilson44 | PS Vita       |
| am_smith | PlayStation 4 |

# 3rd Normal Form

3rd Normal Form - All Fields (columns) can be determined Only by the Key in the table and no other column

| Cust ID  | Cust Name   | Shipping Address     | Newsletter       |
|----------|-------------|----------------------|------------------|
| at_smith | Alan Smith  | 35 Palm St, Miami    | Xbox News        |
| roger25  | Roger Banks | 45 Campus Rd, Boston | PlayStation News |
| wilson44 | Evan Wilson | 28 Rock Av, Denver   | Xbox News        |
| wilson44 | Evan Wilson | 28 Rock Av, Denver   | PlayStation News |
| am_smith | Alan Smith  | 47 Campus Rd, Boston | PlayStation News |

| Primary Key   | Foreign Key |       |
|---------------|-------------|-------|
| Item          | Supplier    | Price |
| Xbox One      | Microsoft   | 250   |
| PlayStation 4 | Sony        | 300   |
| PS Vita       | Sony        | 200   |

| Primary Key | Primary Key   |
|-------------|---------------|
| Cust ID     | Item          |
| at_smith    | Xbox One      |
| roger25     | PlayStation 4 |
| wilson44    | Xbox One      |
| wilson44    | PS Vita       |
| am_smith    | PlayStation 4 |

| Primary Key | Supplier Phone |
|-------------|----------------|
| Microsoft   | (800) BUY-XBOX |
| Sony        | (800) BUY-SONY |

# 4<sup>th</sup> Normal Form

4th Normal Form - No multi-valued dependencies

*Customer Table*

Primary Key

| Cust ID  | Cust Name   | Shipping Address     |
|----------|-------------|----------------------|
| at_smith | Alan Smith  | 35 Palm St, Miami    |
| roger25  | Roger Banks | 47 Campus Rd, Boston |
| wilson44 | Evan Wilson | 28 Rock Av, Denver   |
| am_smith | Alan Smith  | 47 Campus Rd, Boston |

1

1

*Subscription Table*

Primary Key

| Cust ID  | Newsletter       |
|----------|------------------|
| at_smith | Xbox News        |
| roger25  | PlayStation News |
| wilson44 | Xbox News        |
| wilson44 | PlayStation News |
| am_smith | PlayStation News |

N

*Sales Invoice Table*

Primary Key Primary Key

| Cust ID  | Item          |
|----------|---------------|
| at_smith | Xbox One      |
| roger25  | PlayStation 4 |
| wilson44 | Xbox One      |
| wilson44 | PS Vita       |
| am_smith | PlayStation 4 |

N

*Item Table*

Primary Key

Foreign Key

| Item          | Supplier  | Price |
|---------------|-----------|-------|
| Xbox One      | Microsoft | 250   |
| PlayStation 4 | Sony      | 300   |
| PS Vita       | Sony      | 200   |

1

N

*Supplier Table*

Primary Key

| Supplier  | Supplier Phone |
|-----------|----------------|
| Microsoft | (800) BUY-XBOX |
| Sony      | (800) BUY-SONY |

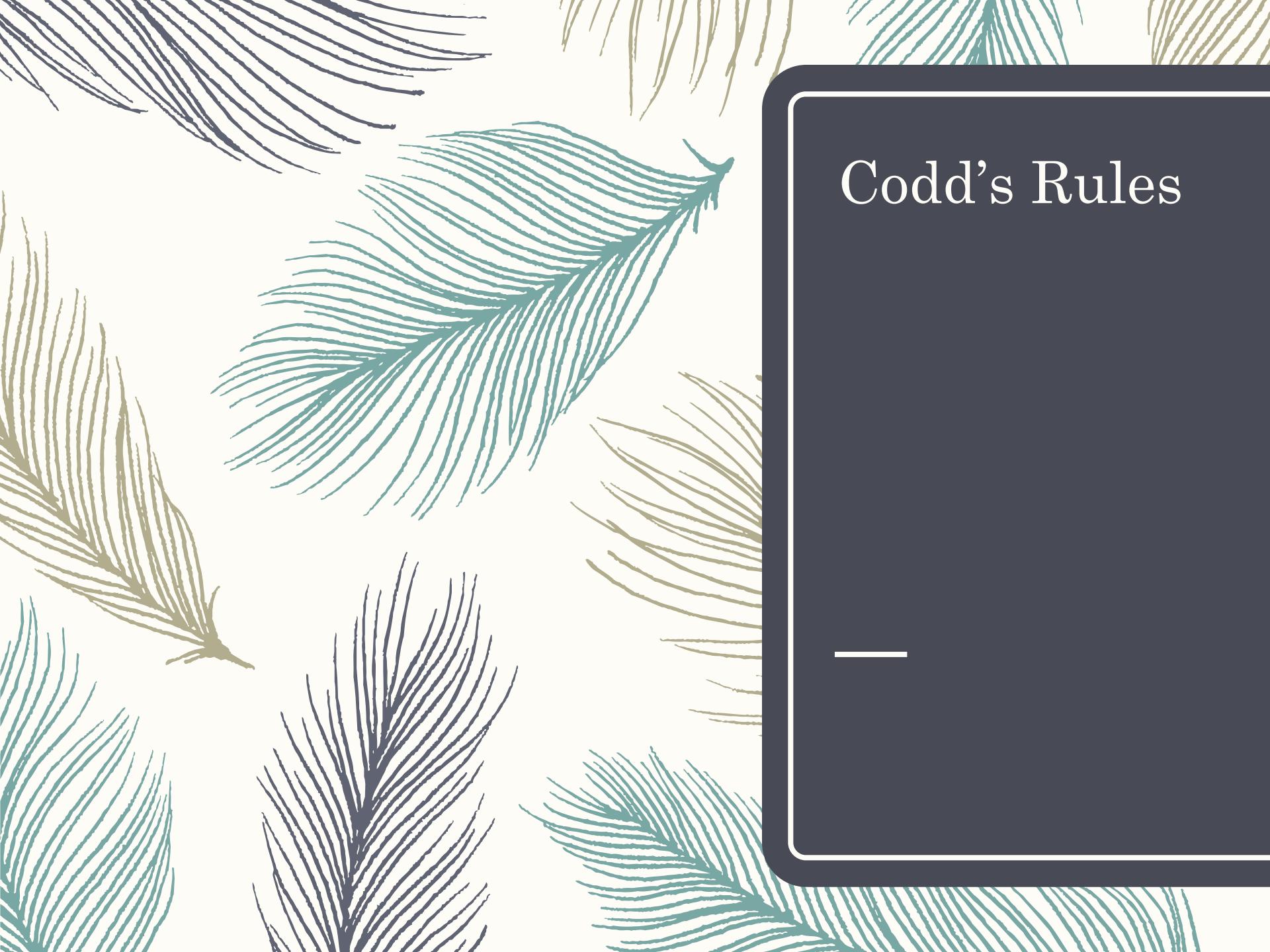
N

1

# References

---

- <https://www.youtube.com/watch?v=UrYLYV7WSHM>



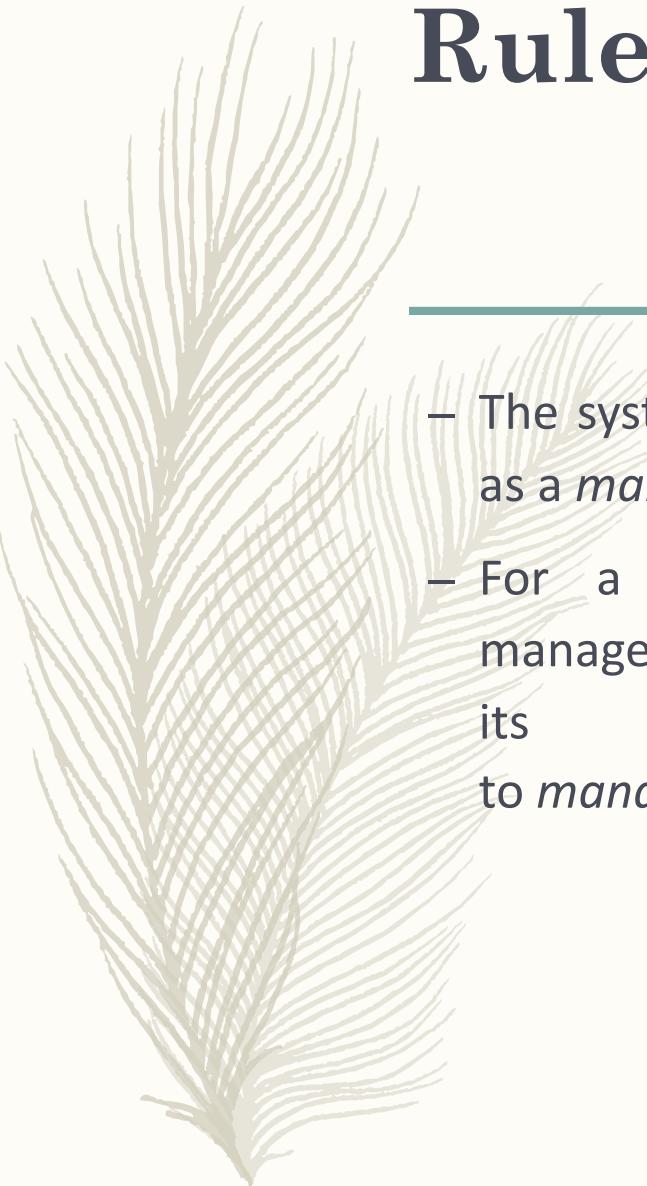
# Codd's Rules

—

# Introduction

---

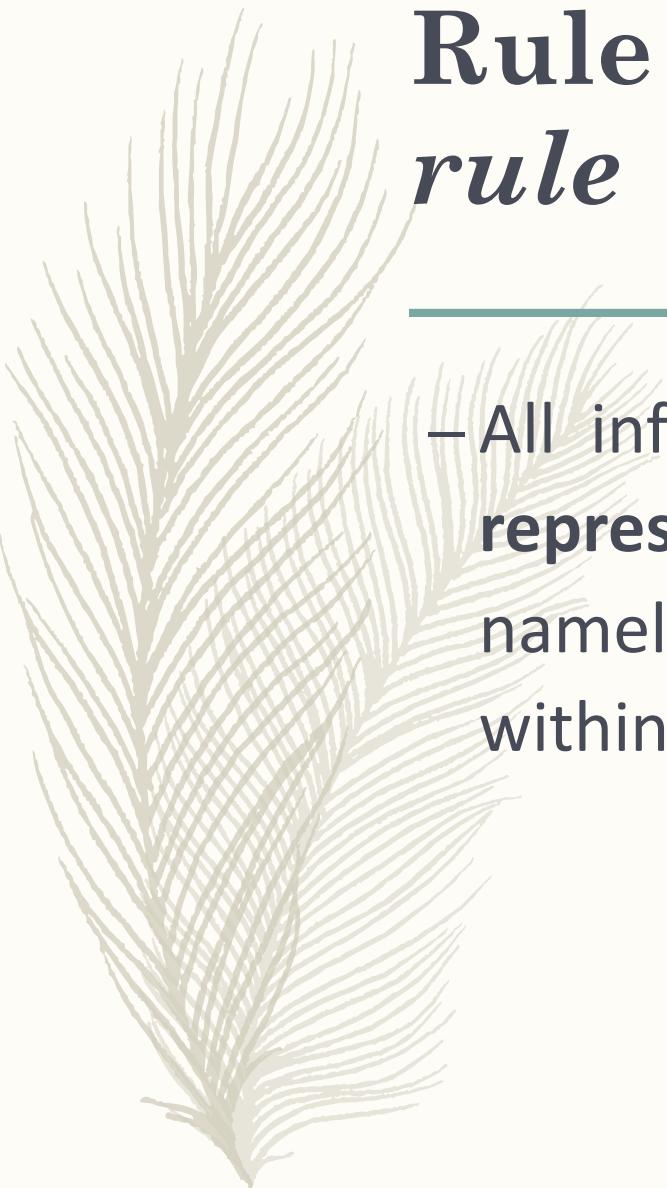
- According to Elmasri and Navathe (1994), Dr. E. F. Codd, the originator of the relational data model, published a two-part article in *Computer World* (Codd, 1985) that lists 12 rules for how to determine whether a DBMS is relational and to what extent it is relational.
- These rules provide a very useful yardstick for evaluating a relational system.



# Rule 0

---

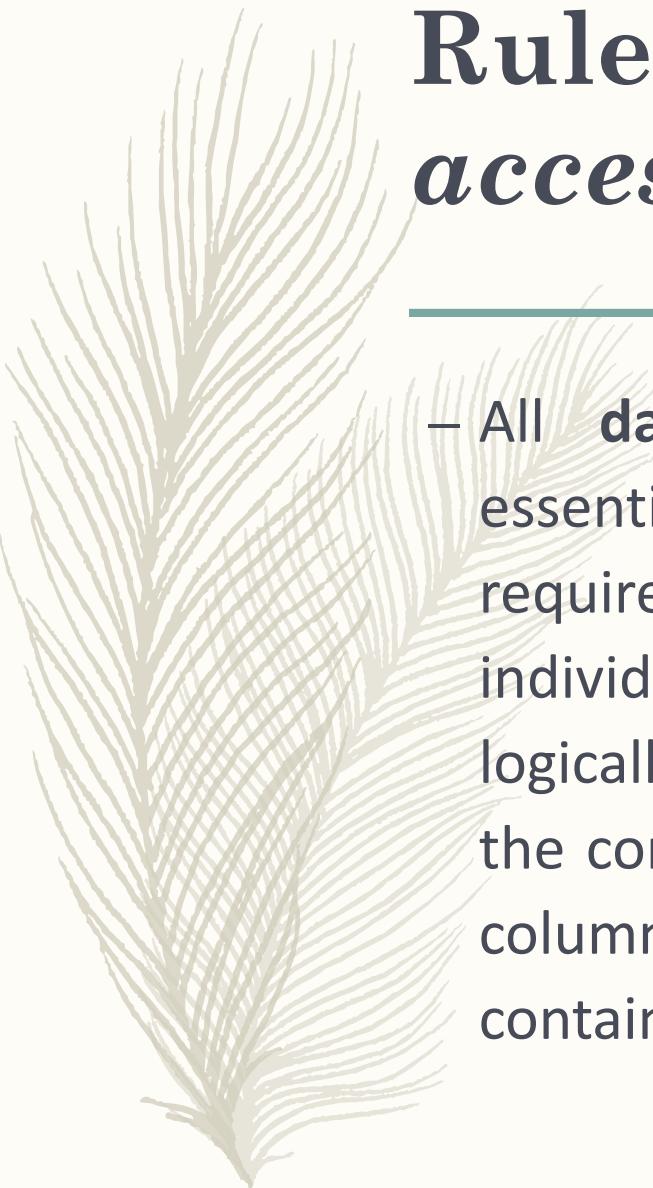
- The system must **qualify as *relational***, as a *database*, and as a *management system*.
- For a system to qualify as a relational database management system (RDBMS), that system must use its *relational* facilities (exclusively) to *manage the database*.



# Rule 1: The *information rule*

---

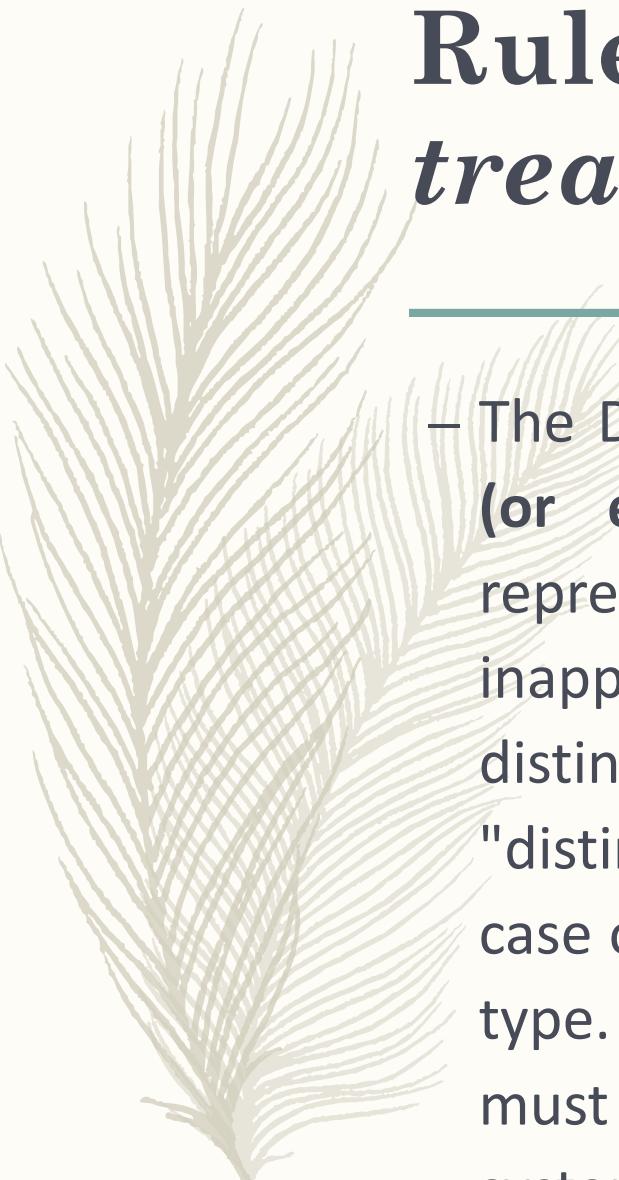
- All information in the database is to be **represented in one and only one way**, namely by values in column positions within rows of tables.



## Rule 2: The *guaranteed access rule*

---

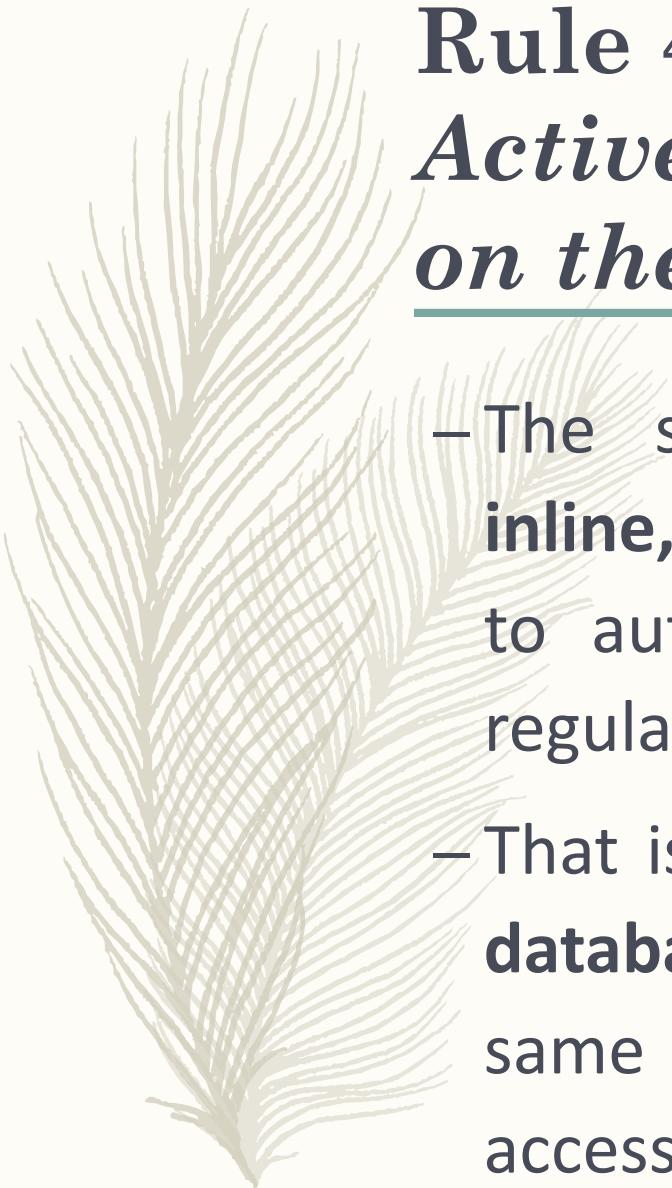
- All **data must be accessible**. This rule is essentially a restatement of the fundamental requirement for primary keys. It says that every individual scalar value in the database must be logically addressable by specifying the name of the containing table, the name of the containing column and the primary key value of the containing row



# *Rule 3: Systematic treatment of null values*

---

- The DBMS must allow each field to remain null (or empty). Specifically, it must support a representation of "missing information and inapplicable information" that is systematic, distinct from all regular values (for example, "distinct from zero or any other number", in the case of numeric values), and independent of data type. It is also implied that such representations must be manipulated by the DBMS in a systematic way

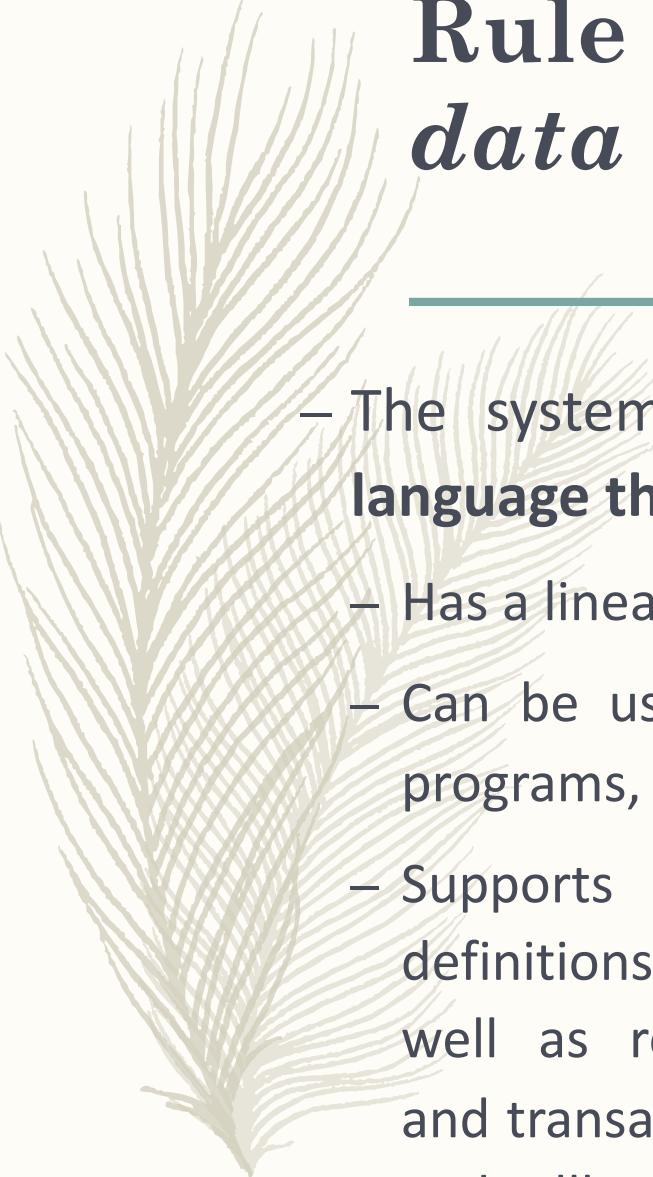


## Rule 4:

### *Active online catalog based on the relational model*

---

- The system must support an **online, inline, relational catalog** that is accessible to authorized users by means of their regular query language.
- That is, users must be able to access the **database's structure (catalog)** using the same query language that they use to access the database's data



## Rule 5: The *comprehensive data sublanguage rule*

---

- The system must support at least one relational language that
  - Has a linear syntax
  - Can be used both interactively and within application programs,
  - Supports data definition operations (including view definitions), data manipulation operations (update as well as retrieval), security and integrity constraints, and transaction management operations (begin, commit, and rollback).

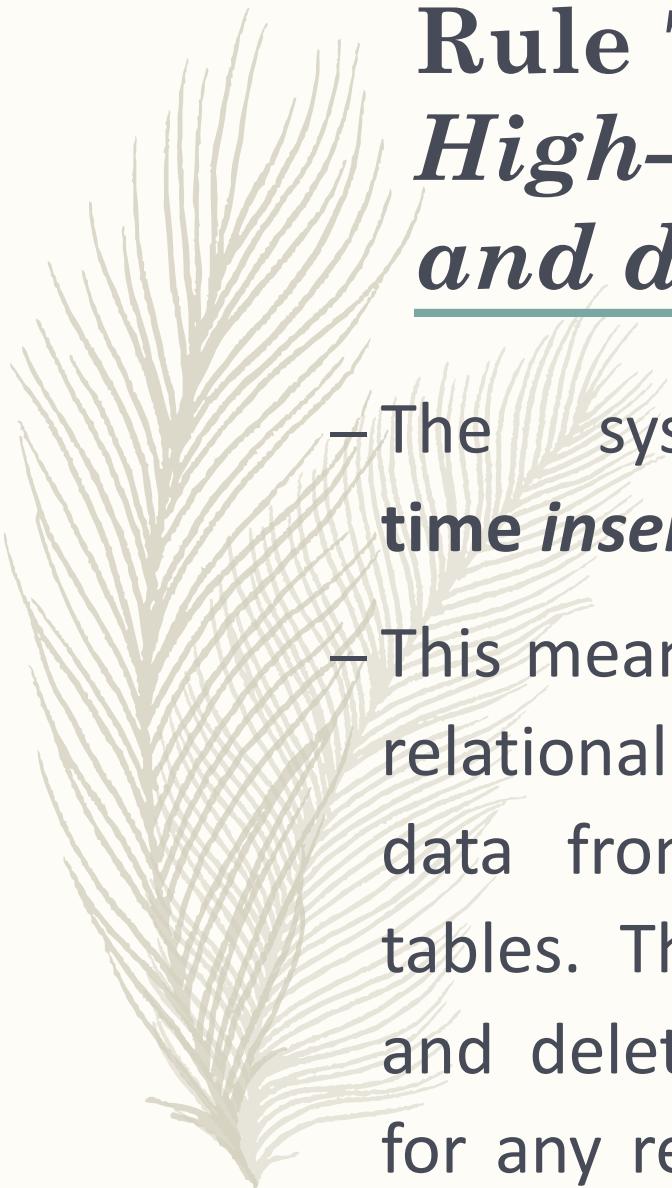


# Rule 6:

## *The view updating rule*

---

- All views that are theoretically updatable must be updatable by the system.



## Rule 7: *High-level insert, update, and delete*

---

- The system must support set-at-a-time *insert, update, and delete* operators.
- This means that data can be retrieved from a relational database in sets constructed of data from multiple rows and/or multiple tables. This rule states that insert, update, and delete operations should be supported for any retrievable set rather than just for a single row in a single table

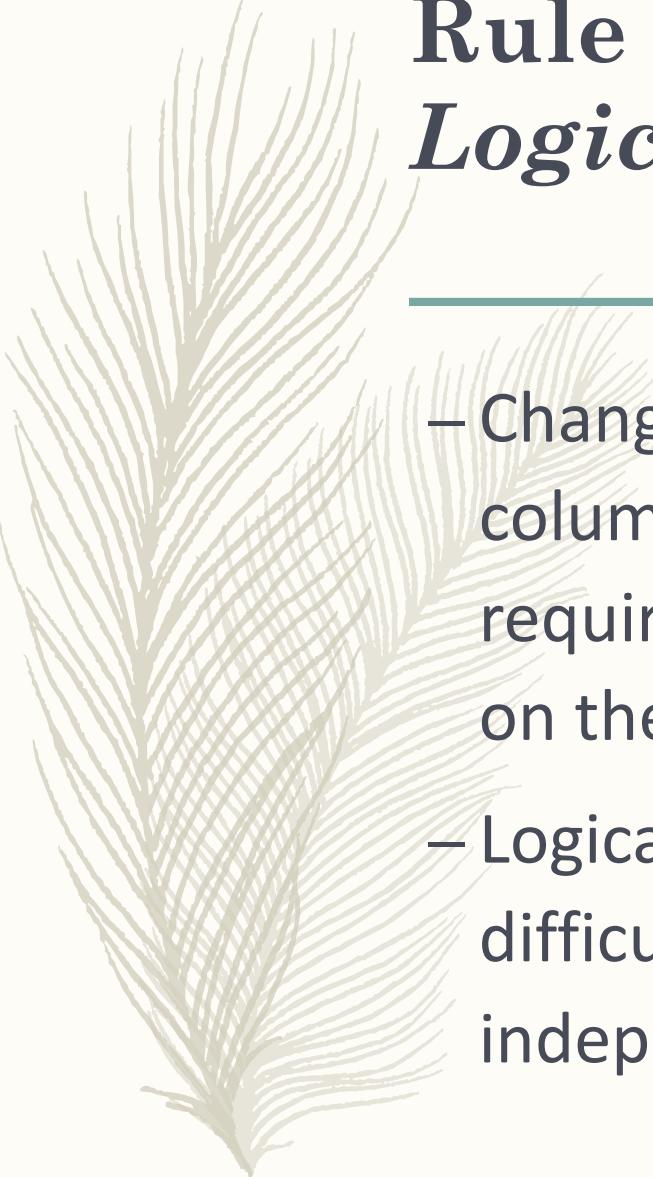


## Rule 8:

### *Physical data independence*

---

- Changes to the physical level (how the data is stored, whether in arrays or linked lists etc.) must not require a change to an application based on the structure



## Rule 9:

### *Logical data independence*

---

- Changes to the logical level (tables, columns, rows, and so on) must not require a change to an application based on the structure.
- Logical data independence is more difficult to achieve than physical data independence

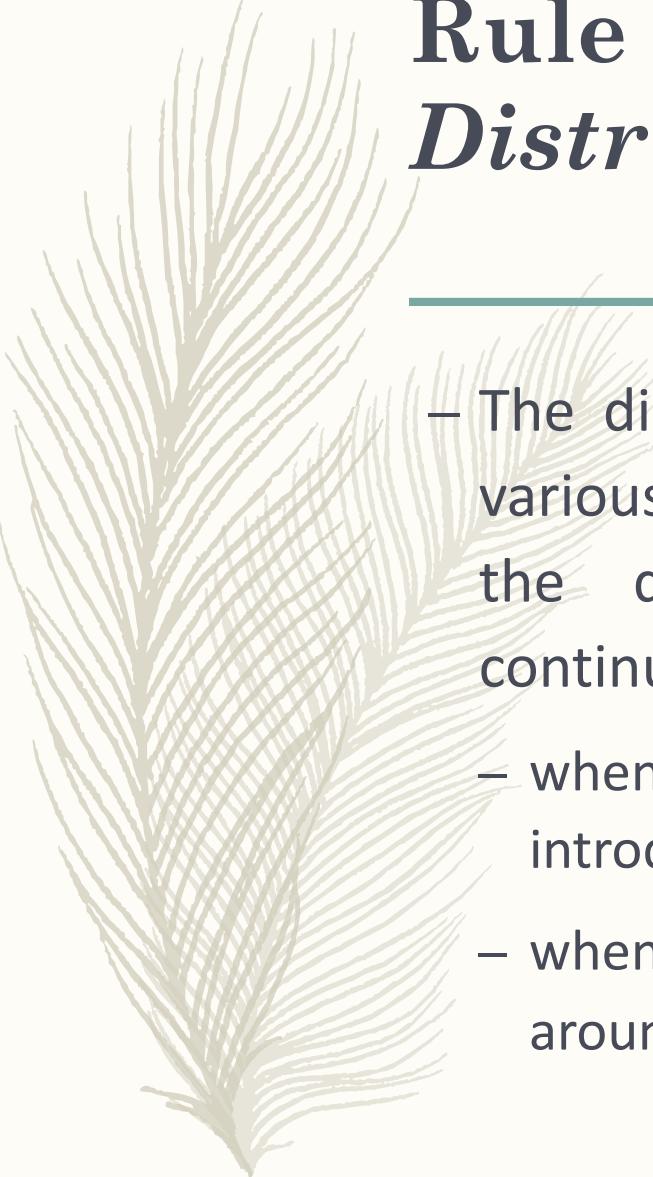


# Rule 10:

## *Integrity independence*

---

- **Integrity constraints** must be specified separately from application programs and **stored in the catalog**.
- It must be possible to change such constraints as and when appropriate without unnecessarily affecting existing applications

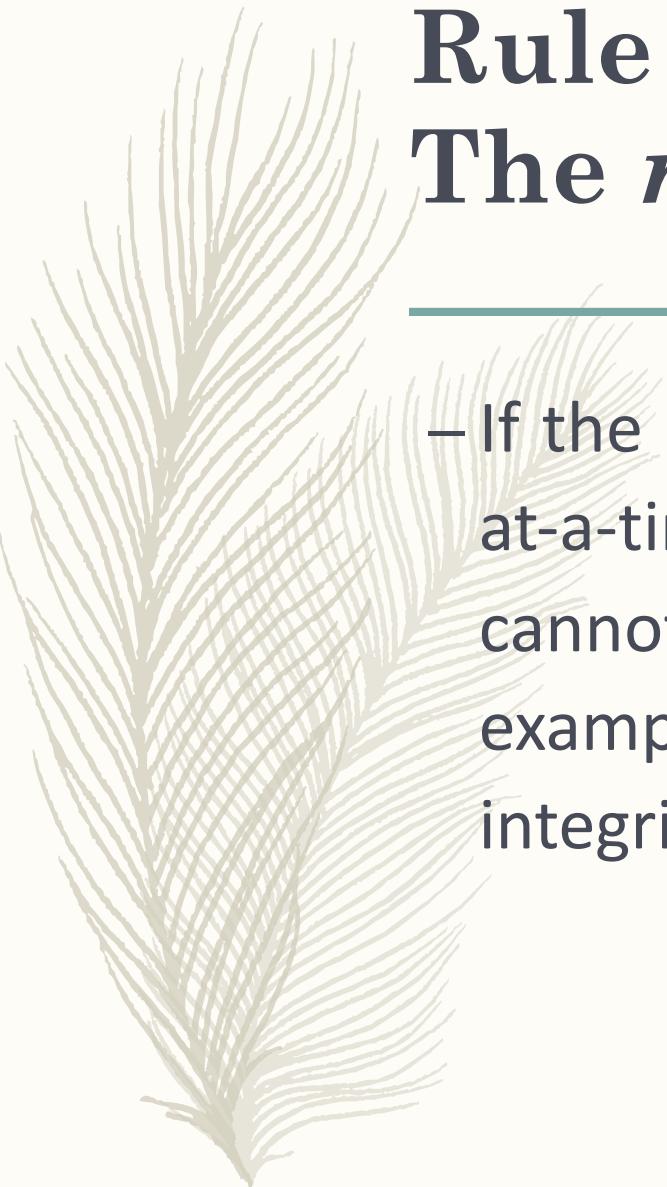


## Rule 11:

### *Distribution independence*

---

- The distribution of portions of the database to various locations should be invisible to users of the database. Existing applications should continue to operate successfully:
- when a distributed version of the DBMS is first introduced; and
- when existing distributed data are redistributed around the system.



## Rule 12: The *nonsubversion rule*

---

- If the system provides a low-level (record-at-a-time) interface, then that interface cannot be used to subvert the system, for example, bypassing a relational security or integrity constraint