



**CHANDIGARH
UNIVERSITY**

Discover. Learn. Empower.

UNIVERSITY INSTITUTE OF ENGINEERING

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

Bachelor of Engineering (Computer Science & Engineering)

Database Management System and CST-227

Prepared By: Ms. Heena Arora

DATABASE MANAGEMENT SYSTEM

DISCOVER . **LEARN** . EMPOWER

Database Management System

Course Outcome

CO Number	Title	Level
CO1	Learn the fundamentals of database systems design and draw ER diagram for the real world problems.	Understand
CO2	Design and query database using SQL.	Apply
CO3	Analyze and apply concepts of normalization to relational database design	Understand
CO4	Learn the concept of transaction, concurrency and recovery.	Remember

Functional Dependencies and Normalization

Functional dependencies

Decomposition

Dependencies-Full Functional Dependency (FFD), Transitive Dependency (TD), Join Dependency (JD), Multi-valued Dependency (MVD)

Normal Forms (1NF, 2NF, 3NF, BCNF)

De-normalization

Functional Dependence (FD)

- A functional dependency is an association between two attributes of the same relational database table.
- One of the attributes is called the determinant and the other attribute is called the determined.
- For each value of the determinant there is associated one and only one value of the determined.

Functional Dependencies

- Functional dependencies (FDs) are used to specify formal measures of the "goodness" of relational designs.
- FDs and keys are used to define **normal forms** for relations
- FDs are **constraints** that are derived from the meaning and interrelationships of the data attributes
- A set of attributes X functionally determines a set of attributes Y if the value of X determines a unique value for Y
- $X \rightarrow Y$ holds if whenever two tuples have the same value for X , they must have the same value for Y
 - If $t1[X]=t2[X]$, then $t1[Y]=t2[Y]$ in any relation instance $r(R)$
- $X \rightarrow Y$ in R specifies a constraint on all relation instances $r(R)$
- If K is a key of R , then K functionally determines all attributes in R (since we never have two distinct tuples with $t1[K]=t2[K]$)
- FDs are derived from the real-world constraints on the attributes

Fully Functional Dependence (FFD)

- Fully Functional Dependence (FFD) is defined, as Attribute Y is FFD on attribute" X, if it is FD on X and not FD on any proper subset of X.

Partial Dependency

- If any proper subsets of the key determine any of the non-key attributes then there exist a partial dependency.
- Example: Given a relation $R(A,B,C,D,E)$, Functional Dependency : $AB \rightarrow CDE$, Primary_key(or simply 'key') is AB.

$A \rightarrow C$: is a Partial Dependency

$A \rightarrow D$: is a Partial Dependency

$A \rightarrow E$: is a Partial Dependency

$B \rightarrow C$: is a Partial Dependency

$B \rightarrow D$: is a Partial Dependency

$B \rightarrow E$: is a Partial Dependency

Transitive Dependency

- The transitivity rule is perhaps the most important one. It states that if X functionally determines Y and Y functionally determine Z then X functionally determines Z .
- $X \rightarrow Y$
- $Y \rightarrow Z$
- $X \rightarrow Z$

Multi-Value Dependency

- A **Multi-Value Dependency (MVD)** occurs when two or more independent multi valued facts about the same attribute occur within the same table.
- **Multivalued dependencies** occur when the presence of one or more rows in a table implies the presence of one or more other rows in that same table.
- **Examples:** For example, imagine a car company that manufactures many models of car, but always makes both red and blue colors of each model. If you have a table that contains the model name, color and year of each car the company manufactures, there is a multivalued dependency in that table.
- If there is a row for a certain model name and year in blue, there must also be a similar row corresponding to the red version of that same car.

Join Dependency

- A **Join Dependency** exists if a relation R is equal to the join of the projections XZ .
- A **join dependency** is a constraint on the set of legal relations over a database scheme. A table T is subject to a join dependency if T can always be recreated by joining multiple tables each having a subset of the attributes of T . If one of the tables in the join has all the attributes of the table T , the join dependency is called trivial.

Inference Rules for FDs

- Given a set of FDs F , we can *infer* additional FDs that hold whenever the FDs in F hold
- Armstrong's inference rules
 - A1. (Reflexive) If Y subset-of X , then $X \rightarrow Y$
 - A2. (Augmentation) If $X \rightarrow Y$, then $XZ \rightarrow YZ$
(Notation: XZ stands for $X \cup Z$)
 - A3. (Transitive) If $X \rightarrow Y$ and $Y \rightarrow Z$, then $X \rightarrow Z$

A1, A2, A3 form a *sound* and *complete* set of inference rules

Additional Useful Inference Rules

- Decomposition
 - If $X \rightarrow YZ$, then $X \rightarrow Y$ and $X \rightarrow Z$
- Union
 - If $X \rightarrow Y$ and $X \rightarrow Z$, then $X \rightarrow YZ$
- Psuedo transitivity
 - If $X \rightarrow Y$ and $WY \rightarrow Z$, then $WX \rightarrow Z$
- **Closure** of a set F of FDs is the set F^+ of all FDs that can be inferred from F

Introduction to Normalization

- **Normalization:** Process of decomposing unsatisfactory "bad" relations by breaking up their attributes into smaller relations
- Normalization is the process of removing redundant data from your tables in order to improve storage efficiency, data integrity and scalability.
- This improvement is balanced against an increase in complexity and potential performance losses from the joining of the normalized tables at query-time.
- There are two goals of the normalization process: eliminating redundant data (for example, storing the same data in more than one table) and ensuring data dependencies make sense (only storing related data in a table). Both of these are worthy goals as they reduce the amount of space a database consumes and ensure that data is logically stored.

Introduction to Normalization

- **Normal form:** Condition using keys and FDs of a relation to certify whether a relation schema is in a particular normal form
 - 2NF, 3NF, BCNF based on keys and FDs of a relation schema
 - 4NF based on keys, multi-valued dependencies
- There is a sequence to normal forms:
 - 1NF is considered the weakest,
 - 2NF is stronger than 1NF,
 - 3NF is stronger than 2NF, and
 - BCNF is considered the strongest
- Also,
 - any relation that is in BCNF, is in 3NF;
 - any relation in 3NF is in 2NF; and
 - any relation in 2NF is in 1NF.

Need of Normalization

- Normalization is the aim of well design Relational Database Management System (RDBMS). It is step by step set of rules by which data is put in its simplest forms. We normalize the relational database management system because of the following reasons:
 - Minimize data redundancy i.e. no unnecessarily duplication of data.
 - To make database structure flexible i.e. it should be possible to add new data values and rows without reorganizing the database structure.
 - Data should be consistent throughout the database i.e. it should not suffer from following anomalies.
 - Complex queries required by the user should be easy to handle.

First Normal Form

- Disallows composite attributes, multivalued attributes, and nested relations; attributes whose values *for an individual tuple* are non-atomic.
- We say a relation is in 1NF if all values stored in the relation are single-valued and atomic.

First Normal Form : Example

- An outer join between Employee and Employee Degree will produce the information we saw before

<u>EmpNum</u>	EmpPhone	EmpDegrees
123	233-9876	
333	233-1231	BA, BSc, PhD
679	233-1231	BSc, MSc

Employee

EmpNum	EmpPhone
123	233-9876
333	233-1231
679	233-1231

EmployeeDegree

EmpNum	EmpDegree
333	BA
333	BSc
333	PhD
679	BSc
679	MSc

Figure -1: Example of First Normal Form

Second Normal Form

- A relation is in **2NF** if it is in 1NF, and every non-key attribute is fully dependent on each candidate key. (That is, we don't have any partial functional dependency.)
- 2NF (and 3NF) both involve the concepts of key and non-key attributes.
- A *key attribute* is any attribute that is part of a key; any attribute that is not a key attribute, is a *non-key attribute*

Third Normal Form

- A database is in third normal form if it satisfies the following conditions:
 - It is in second normal form
 - There is no transitive functional dependency
- Definition
 - **Transitive functional dependency** – if there a set of attribute Z that are neither a primary or candidate key and both $X \rightarrow Z$ and $Y \rightarrow Z$ holds.

Third Normal Form : Example

- In the table able, [Book ID] determines [Genre ID], and [Genre ID] determines [Genre Type]. Therefore, [Book ID] determines [Genre Type] via [Genre ID] and we have transitive functional dependency, and this structure does not satisfy third normal form.
- To bring this table to third normal form, we split the table into two as follows:

TABLE_BOOK_DETAIL

Book ID	Genre ID	Genre Type	Price
1	1	Gardening	25.99
2	2	Sports	14.99
3	1	Gardening	10.00
4	3	Travel	12.99
5	2	Sports	17.99

TABLE_BOOK

Book ID	Genre ID	Price
1	1	25.99
2	2	14.99
3	1	10.00
4	3	12.99
5	2	17.99

TABLE_GENRE

Genre ID	Genre Type
1	Gardening
2	Sports
3	Travel

Figure -2: Example of Third Normal Form

Now all non-key attributes are fully functional dependent only on the primary key. In [TABLE_BOOK], both [Genre ID] and [Price] are only dependent on [Book ID]. In [TABLE_GENRE], [Genre Type] is only dependent on [Genre ID].

BCNF (Boyce- Codd Normal Form)

- A relation schema R is in **Boyce-Codd Normal Form (BCNF)** if whenever
- an FD $X \rightarrow A$ holds in R, then X is a Super key of R
- Each normal form is strictly stronger than the previous one:
 - Every 2NF relation is in 1NF
 - Every 3NF relation is in 2NF
 - Every BCNF relation is in 3NF
- There exist relations that are in 3NF but not in BCNF
- The goal is to have each relation in BCNF (or 3NF)

BCNF (Boyce- Codd Normal Form)

- For a table to be in BCNF, following conditions must be satisfied:
- R must be in 3rd Normal Form
- and, for each functional dependency ($X \rightarrow Y$), X should be a super Key.

Denormalization

- Causes redundancy, but fast performance & no referential integrity
- Denormalize when
 - specific queries occur frequently,
 - a strict performance is required and
 - it is not heavily updated
- So, de normalize only when there is a very clear advantage to doing so and document carefully the reason for doing so

Problems of Denormalization

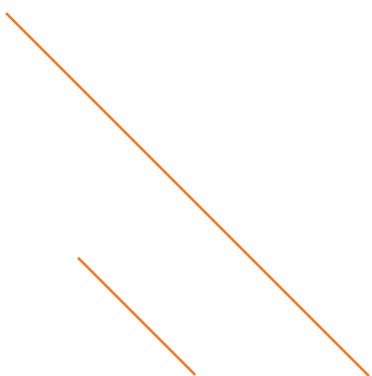
- Makes row longer
- Makes data transfer longer
- Needs more memory for memory processing
- Cause redundancy and expensive update

REFERENCES

- “Data Structures and Algorithms Made Easy: Data Structures and Algorithmic Puzzles” by Narasimha Karumanchi
- “Data Structures Using C” by Aaron M. Tenenbaum
- “Data Structures and Algorithms” by Alfred V. Aho
- “Fundamentals of Data Structures in C” by Ellis Horowitz, Sartaj Sahni, and Susan Anderson-Freed

A large, stylized white graphic on the left side of the slide, composed of multiple parallel lines forming a large, open 'L' or 'C' shape.

THANK YOU

Two thin, parallel orange diagonal lines in the top-right corner of the slide.Two thin, parallel orange diagonal lines in the bottom-left corner of the slide.

For queries
Email: Heenae7725@cumail.in