Normalization

Normalization | Introduction

- **Definition:** Normalization is a database design technique that reduces data redundancy to eliminates Insertion, Update and Deletion Anomalies.
- **Procedure:** Divides larger tables into smaller tables and links them using relationships.
- **Purpose:** Eliminate redundant data and ensure data is stored logically.
- **Stage:** At the level of logical model of database.
- **Inventor:** Edgar Codd proposed the theory of normalization of data. Later he joined Raymond F. Boyce to develop the the Boyce-Codd Normal Form.

Normalization | Types of Dependencies

- **Definition:** A constraint that defines the relationship between attributes.

- **Representation:** It can be represented as A->B. Here B is called as dependent and A is called as determinant.

- **Types of dependencies:**
  - **Functional dependency (FD):** A relation R contains A,B,C and D attributes, if B attribute is depend on A then the dependency is called functional dependency.

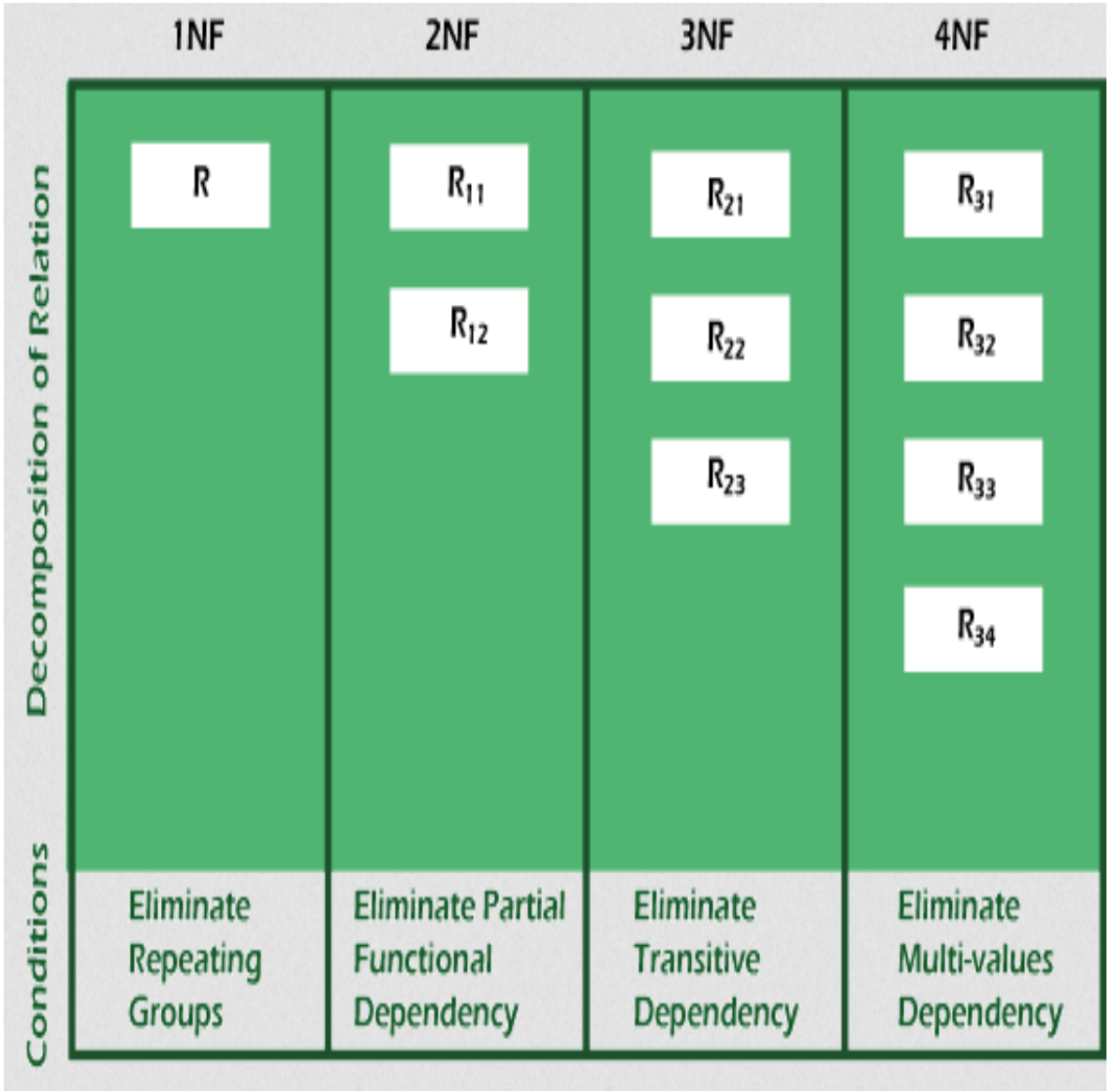
  - **Partial Functional Dependency (PFD):** A relation R contains A,B,C and D attributes and A & B are Composite Primary Key attributes, if C is dependent only on A not depend on B called PFD.

Normalization | Types of Dependencies

- **Types of dependencies:**
  - **Transitive Functional Dependency (TFD):** If A, B, and C are attributes of a relation such that if A-> B and B -> C, then C is transitively dependent on A.
  - **Multi-Valued Dependency (MVD):** Multivalued dependency occurs when there are more than one independent multivalued attributes in a table.
  - **Full Functional Dependency (FFD):** A relation R contains A,B,C and D attributes and A & B are Composite Primary Key attributes, if C attribute is depend on both A and B then the dependency is called FFD.

Normalization | Types

- First Normal Form [1NF]
- Second Normal Form [2NF]
- Third Normal Form [3NF]
- Boyce – Codd Normal Form [BCNF]
- Fourth Normal Form [4NF]



Normalization | Un-Normalized Data

SID	NAME	CONTACTNO	COURSE	MARKS	GRADE
1	ROHAN	111-111-1111, 123-456-7890	OOPS	80	B+
1	ROHAN	111-111-1111, 123-456-7890	DBMS	95	A+
2	RAVI	222-222-3222	OOPS	75	B
2	RAVI	222-222-3222	DWDM	85	A
3	NEHA	333-333-3333	OOPS	90	A+

Normalization | First Normal Form [1NF]

- A relation R is in 1NF if and only if each attribute contains atomic values

- This will remove repeating groups in a table.

SID	NAME	CONTACTNO1	CONTACTNO2	COURSE	MARKS	GRADE
1	ROHAN	111-111-1111	123-456-7890	OOPS	80	B+
1	ROHAN	111-111-1111	123-456-7890	DBMS	95	A+
2	RAVI	222-222-3222	NULL	OOPS	75	B
2	RAVI	222-222-3222	NULL	DWDM	85	A
3	NEHA	333-333-3333	NULL	OOPS	90	A+

- The composite candidate key for this relation will be SId and Course as SId alone is not unique.

Normalization | Second Normal Form [2NF]

- A relation is in second normal form if and only if
  - R is already in 1NF and there is no partial dependency between non-key attributes and key attributes.

SID	NAME	CONTACTNO1	CONTACTNO2	COURSE	MARKS	GRADE
1	ROHAN	111-111-1111	123-456-7890	OOPS	80	B+
1	ROHAN	111-111-1111	123-456-7890	DBMS	95	A+
2	RAVI	222-222-3222	NULL	OOPS	75	B
2	RAVI	222-222-3222	NULL	DWDM	85	A
3	NEHA	333-333-3333	NULL	OOPS	90	A+

- If we carefully observe the above relation we can find that there are four attributes are dependent on partial candidate key.
  - SID, COURSE -> NAME, CONTACTNO1, CONTACTNO2, GRADE
- To make this relation 2NF compliant we need to remove this partial dependency and decompose the relation.

Normalization | Second Normal Form [2NF]

- Student Table:

SID	NAME	CONTACT NO1	CONTACT NO2
1	ROHAN	111-111-1111	123-456-7890
2	RAVI	222-222-3222	NULL
3	NEHA	333-333-3333	NULL

- Marks Table:

SID	COURSE	MARKS	GRADE
1	OOPS	80	B+
1	DBMS	95	A+
2	OOPS	75	B
2	DWDM	85	A
3	OOPS	90	A+

Normalization | Third Normal Form [3NF]

- A relation R is said to be in the Third Normal Form (3NF) if and only if :
  - It is in 2NF and
  - Transitive dependency does not exists between key attributes and non-key attributes through another non-key attribute.

- Let us look at the following dependencies
  - SID, COURSE -> MARKS
  - MARKS -> GRADE
  - STUDENTID, COURSE -> GRADE

SID	COURSE	MARKS	GRADE
1	OOPS	80	B+
1	DBMS	95	A+
2	OOPS	75	B
2	DWDM	85	A
3	OOPS	90	A+

Normalization | Third Normal Form [3NF]

Marks Table:

SID	COURSE	MARKS	GRADE
1	OOPS	80	B+
1	DBMS	95	A+
2	OOPS	75	B
2	DWDM	85	A
3	OOPS	90	A+

Marks Table:

SID	COURSE	MARKS
1	OOPS	80
1	DBMS	95
2	OOPS	75
2	DWDM	85
3	OOPS	90

Grade Table:

LOWMARKS	HIGHMARKS	GRADE
90	100	A+
75	89	A
60	74	B
40	59	C
0	39	F

Normalization | Boyce – Codd Normal Form [BCNF]

- For BCNF the table should be in 3NF and every FD, LHS is super key.
- A table is in BCNF if every functional dependency X → Y, X is the super key of the table.
- **Example:**

EMP_ID	EMP_COUNTRY	EMP_DEPT	DEPT_TYPE	EMP_DEPT_NO
264	India	Designing	D394	283
264	India	Testing	D394	300
364	UK	Stores	D283	232
364	UK	Developing	D283	549

- In the above table Functional dependencies are as follows:
  - EMP_ID → EMP_COUNTRY
  - EMP_DEPT → {DEPT_TYPE, EMP_DEPT_NO}

Normalization | Boyce – Codd Normal Form [BCNF]

Country Table:

EMP_ID	EMP_COUNTRY
264	India
264	India
364	UK
364	UK

Functional Dependencies:

- EMP_ID → EMP_COUNTRY
- EMP_DEPT → {DEPT_TYPE, EMP_DEPT_NO}

Dept Table:

EMP_DEPT	DEPT_TYPE	EMP_DEPT_NO
Designing	D394	283
Testing	D394	300
Stores	D283	232
Developing	D283	549

EMP_Dept_Mapping Table:

EMP_ID	EMP_DEPT_NO
264	283
264	300
364	232
364	549

Normalization | Fourth Normal Form [4NF]

- If a table is in 4NF that should satisfy the following two conditions:
  - It should be in the Boyce-Codd Normal Form [BCNF].
  - The table should not have any Multi-valued Dependency
- Subject Table:

COURSE	STUDENT	BOOKS
DBMS	ROHAN	DATA BASE MANAGEMENT SYSTEMS
DBMS	NEHA	DATA BASE SYSTEMS CONCEPTS
OOPS	ROHAN	JAVA COMPLETE REFERENCE
OOPS	RAVI	HEAD FIRST JAVA
OOPS	RAVI	FUNDAMENTALS OF OOPS
OOPS	NEHA	HEAD FIRST JAVA

- MVD can be represent as: COURSE ->->STUDENT and COURSE ->->BOOKS

Normalization | Fourth Normal Form [4NF]

CourseStudent Table:

COURSE	STUDENT
DBMS	ROHAN
DBMS	NEHA
OOPS	ROHAN
OOPS	RAVI
OOPS	RAVI
OOPS	NEHA

CourseBooks Table:

COURSE	BOOKS
DBMS	DATA BASE MANAGEMENT SYSTEMS
DBMS	DATA BASE SYSTEMS CONCEPTS
OOPS	JAVA COMPLETE REFERENCE
OOPS	HEAD FIRST JAVA
OOPS	FUNDAMENTALS OF OOPS
OOPS	HEAD FIRST JAVA

Transactions
And
Concurrency Control

Transactions and Concurrency Control

- A set of logically related operations is known as transaction and allows you to combine multiple operations into a single unit of work.
- A transaction consists of a single command or a group of commands.
- If a failure occurs at one point in the transaction, all of the updates can be rolled back to their pre-transaction state.
- Most relational database systems support transactions by providing locking, logging, and transaction management facilities whenever a client application performs an update, insert, or delete operation.
- Concurrency Control deals with interleaved execution of more than one transaction.

Transactions and Concurrency Control

- In a single-user database, the user can modify data in the database without concern for other users modifying the same data at the same time. However, in a multiuser database, the statements within multiple simultaneous transactions can update the same data.
- Transactions executing at the same time need to produce meaningful and consistent results. Therefore, control of data concurrency and data consistency is vital in a multiuser database.
- **Data concurrency** means that many users can access data at the same time.
- **Data consistency** means that each user sees a consistent view of the data, including visible changes made by the user's own transactions and transactions of other users.
- A transaction must conform to the **ACID** properties—atomicity, consistency, isolation, and durability—in order to guarantee data consistency.

Transaction | Properties

- The main operations of a transaction are:
  - **Read:** Read operations Read(A) or R(A) reads the value of A from the database and stores it in a buffer in main memory.
  - **Write:** Write operation Write(A) or W(A) writes the value back to the database from buffer.

- **Example:** Let us take a debit transaction from an account which consists of following three operations:
  1. R(A);
  2. A=A-1000;
  3. W(A);

Transaction | ACID Properties | Atomicity

Atomicity: As a transaction is set of logically related operations, either all of them should be executed or none.

Example:

- Assume A's value before starting of transaction is 5000.
  1. The first operation reads the value of A from database and stores it in a buffer.
  2. Second operation will decrease its value by 1000. So buffer will contain 4000.
  3. Third operation will write the value from buffer to database. So A's final value will be 4000.
- But it may also be possible that transaction may fail after executing some of its operations. The failure can be because of hardware, software or power etc.
- If debit transaction fails after executing operation 1 and 2 then its new value 4000 will not be updated in the database which leads to inconsistency

Transaction | ACID Properties | Consistency

Consistency: If operations of debit and credit transactions on same account are executed concurrently, it may leave database in an inconsistent state.

Example:

- T1 (debit of Rs. 1000 from A) and T2 (credit of 500 to A) executing concurrently, the database reaches inconsistent state.
- Let us assume Account balance of A is Rs. 5000. T1 reads A(5000) and stores the value in its local buffer space. Then T2 reads A(5000) and also stores the value in its local buffer space.
- T1 performs A=A-1000 (5000-1000=4000) and 4000 is stored in T1 buffer space. Then T2 performs A=A+500 (5000+500=5500) and 5500 is stored in T2 buffer space. T1 writes the value from its buffer back to database.
- A's value is updated to 4000 in database and then T2 writes the value from its buffer back to database. A's value is updated to 5500 which shows that the effect of debit transaction is lost and database has become inconsistent.

Transaction | ACID Properties | Isolation

Isolation:

- This property ensures that multiple transactions can occur concurrently without leading to inconsistency of database state.
- Result of a transaction should not be visible to others before transaction is committed.

Example:

- Let us assume that A's balance is Rs. 5000 and T1 debits Rs. 1000 from A.
- A's new balance will be 4000. If T2 credits Rs. 500 to A's new balance, A will become 4500 and after this T1 fails.
- Then we have to rollback T2 as well because it is using value produced by T1.
- So a transaction results are not made visible to other transactions before it commits.

Transaction | ACID Properties | Durability

Durable:

- Once database has committed a transaction, the changes made by the transaction should be permanent.

Example:

- If a person has credited Rs. 600000 to his account, bank can't say that the update has been lost.

- To avoid this problem, multiple copies of database are stored at different locations.

Concurrency Control

- Serializability can be called a process used for finding the correct non-serial schedules in the database.
- Serializability helps to maintain the consistency in the database and often relates to the isolation features of a transaction.
- The serializable mode of transaction behavior tries to ensure that transactions to be executed one at a time, or serially, than concurrently.
- In a multiprogramming environment where multiple transactions can be executed simultaneously, it is important to control the concurrency of transactions.
- We have concurrency control protocols to ensure atomicity, isolation, and serializability of concurrent transactions.
- Concurrency control protocols can be broadly divided into two categories
  1. **Lock based protocols**
  2. **Time stamp based protocols**

Concurrency Control

1. Lock based protocols

- Database systems equipped with lock-based protocols use a mechanism by which any transaction cannot read or write data until it acquires an appropriate lock on it

2. Time stamp based protocols

- The most commonly used concurrency protocol is the timestamp based protocol. This protocol uses either system time or logical counter as a timestamp.

- Lock-based protocols manage the order between the conflicting pairs among transactions at the time of execution, whereas timestamp-based protocols start working as soon as a transaction is created.

Thank You!