

Normal Forms Explained



Explain criteria for all normal forms with examples



Criteria for Normal Forms in Relational Databases

Normal forms are standards for database design to minimize redundancy and improve data integrity. Below are the criteria for each normal form with examples:

1. First Normal Form (1NF)

Criteria:

- Each cell in a table must contain a single value (atomic value).
- Each column must contain values of the same data type.
- The table must have a unique identifier (primary key).

Example:

Non-1NF Table:

ID	Name	Hobbies
1	John Smith	Reading, Writing
2	Alice Doe	Drawing, Dancing

1NF Table:

ID	Name	Hobby
1	John Smith	Reading
1	John Smith	Writing
2	Alice Doe	Drawing
2	Alice Doe	Dancing

2. Second Normal Form (2NF)

Criteria:

- The table must be in 1NF.
- All non-prime attributes must depend on the entire primary key (no partial dependency).

Example:

Non-2NF Table:

OrderID	ProductID	ProductName	OrderDate
1	101	Laptop	2024-12-01

OrderID	ProductID	ProductName	OrderDate
2	102	Tablet	2024-12-02

Issue: `ProductName` depends only on `ProductID`, not the entire composite key (`OrderID`, `ProductID`).

2NF Tables:

Orders Table:

OrderID	OrderDate
1	2024-12-01
2	2024-12-02

Products Table:

ProductID	ProductName
101	Laptop
102	Tablet

OrderDetails Table:

OrderID	ProductID
1	101
2	102

3. Third Normal Form (3NF)

Criteria:

- The table must be in 2NF.
- There should be no transitive dependency (non-prime attributes must not depend on other non-prime attributes).

Example:

Non-3NF Table:

EmployeeID	DepartmentID	DepartmentName	Salary
1	10	HR	50000
2	20	IT	60000

Issue: `DepartmentName` depends on `DepartmentID`, not `EmployeeID`.

3NF Tables:

Employees Table:

EmployeeID	DepartmentID	Salary
1	10	50000
2	20	60000

Departments Table:

DepartmentID	DepartmentName
10	HR
20	IT

4. Boyce-Codd Normal Form (BCNF)

Criteria:

- The table must be in 3NF.
- Every determinant must be a candidate key.

Example:

Non-BCNF Table:

StudentID	CourseID	Instructor
1	101	Dr. Smith
2	101	Dr. Smith

Issue: `Instructor` depends on `CourseID`, not the entire primary key (`StudentID`, `CourseID`).

BCNF Tables:

Courses Table:

CourseID	Instructor
101	Dr. Smith

Enrollments Table:

StudentID	CourseID
1	101
2	101

5. Fourth Normal Form (4NF)

Criteria:

- The table must be in BCNF.
- No multi-valued dependencies.

Example:

Non-4NF Table:

StudentID	CourseID	Hobby
1	101	Reading
1	101	Writing

Issue: Multi-valued dependency: `StudentID` → `CourseID` and `StudentID` → `Hobby`.

4NF Tables:

Enrollments Table:

StudentID	CourseID
1	101

Hobbies Table:

StudentID	Hobby
1	Reading
1	Writing

6. Fifth Normal Form (5NF)

Criteria:

- The table must be in 4NF.
- All join dependencies must be lossless.

Example:

Non-5NF Table:

ProjectID	EmployeeID	SkillID
1	101	S1
1	102	S2

Issue: Redundancy due to possible combinations of `ProjectID`, `EmployeeID`, and `SkillID`.

5NF Tables:

Projects Table:

ProjectID	EmployeeID
1	101
1	102

Skills Table:

EmployeeID	SkillID
101	S1
102	S2

6. Domain-Key Normal Form (DKNF)

Criteria:

- The table must have no constraints other than domain and key constraints.

Example:

A table that is already in 5NF and has no complex integrity constraints automatically qualifies as DKNF.

These normalization steps help ensure data consistency and minimize redundancy.



Also explain all the joins



Types of SQL Joins

Joins are used in SQL to combine rows from two or more tables based on a related column between them. Here are the different types of joins, with explanations and examples:

1. INNER JOIN

- **Definition:** Returns rows where there is a match in both tables.
- **Key Point:** Rows without a match in either table are excluded.

Syntax:

```
sql

SELECT column_names
FROM table1
INNER JOIN table2
ON table1.common_column = table2.common_column;
```

Example:

Employees Table:

EmployeeID	Name	DepartmentID
1	John Smith	101
2	Alice Doe	102

Departments Table:

DepartmentID	DepartmentName
101	HR
103	IT

Query:

```
sql

SELECT Employees.Name, Departments.DepartmentName
FROM Employees
INNER JOIN Departments
ON Employees.DepartmentID = Departments.DepartmentID;
```

Result:

Name	DepartmentName
John Smith	HR

2. LEFT JOIN (or LEFT OUTER JOIN)

- **Definition:** Returns all rows from the left table and the matching rows from the right table. If no match is found, `NULL` is returned for columns from the right table.
- **Key Point:** Useful to find rows in the left table without matches in the right table.

Syntax:

```
sql

SELECT column_names
FROM table1
LEFT JOIN table2
ON table1.common_column = table2.common_column;
```

Example:

Query:

```
sql

SELECT Employees.Name, Departments.DepartmentName
FROM Employees
LEFT JOIN Departments
ON Employees.DepartmentID = Departments.DepartmentID;
```

Result:

Name	DepartmentName
John Smith	HR
Alice Doe	NULL

3. RIGHT JOIN (or RIGHT OUTER JOIN)

- **Definition:** Returns all rows from the right table and the matching rows from the left table. If no match is found, `NULL` is returned for columns from the left table.
- **Key Point:** Useful to find rows in the right table without matches in the left table.

Syntax:

```
sql

SELECT column_names
FROM table1
RIGHT JOIN table2
ON table1.common_column = table2.common_column;
```

Example:

Query:

```
sql

SELECT Employees.Name, Departments.DepartmentName
FROM Employees
RIGHT JOIN Departments
ON Employees.DepartmentID = Departments.DepartmentID;
```

Result:

Name	DepartmentName
John Smith	HR
NULL	IT

4. FULL JOIN (or FULL OUTER JOIN)

- **Definition:** Returns rows when there is a match in either table. Rows without a match in one table will have `NULL` for the columns of the other table.
- **Key Point:** Combines `LEFT JOIN` and `RIGHT JOIN`.

Syntax:

```
sql

SELECT column_names
FROM table1
FULL JOIN table2
ON table1.common_column = table2.common_column;
```

Example:

Query:

```
sql

SELECT Employees.Name, Departments.DepartmentName
FROM Employees
FULL JOIN Departments
ON Employees.DepartmentID = Departments.DepartmentID;
```

Result:

Name	DepartmentName
John Smith	HR
Alice Doe	NULL
NULL	IT

5. CROSS JOIN

- **Definition:** Produces a Cartesian product, returning all combinations of rows from both tables.
- **Key Point:** Does not require a matching condition.

Syntax:

```
sql

SELECT column_names
FROM table1
```

```
CROSS JOIN table2;
```

Example:

Query:

```
sql

SELECT Employees.Name, Departments.DepartmentName
FROM Employees
CROSS JOIN Departments;
```

Result:

Name	DepartmentName
John Smith	HR
John Smith	IT
Alice Doe	HR
Alice Doe	IT

6. SELF JOIN

- **Definition:** Joins a table to itself, treating the table as two separate tables (using aliases).
- **Key Point:** Useful for hierarchical or recursive relationships within a table.

Syntax:

```
sql

SELECT a.column_name, b.column_name
FROM table_name a
INNER JOIN table_name b
ON a.common_column = b.common_column;
```

Example:

Employees Table:

EmployeeID	Name	ManagerID
1	John Smith	NULL
2	Alice Doe	1
3	Bob Brown	1

Query:

```
sql

SELECT e1.Name AS Employee, e2.Name AS Manager
FROM Employees e1
LEFT JOIN Employees e2
ON e1.ManagerID = e2.EmployeeID;
```


Result:

Employee	Manager
John Smith	NULL
Alice Doe	John Smith
Bob Brown	John Smith

7. NATURAL JOIN

- **Definition:** Automatically joins tables based on all columns with the same name and compatible data types.
- **Key Point:** No `ON` clause is needed.

Syntax:

```
sql

SELECT column_names
FROM table1
NATURAL JOIN table2;
```

Example:

Query:

```
sql

SELECT *
FROM Employees
NATURAL JOIN Departments;
```

Result: Combines rows based on matching `DepartmentID`.

Each join type is used based on the relationship and requirements of your query.



canonical cover with example



Canonical Cover in DBMS

A **canonical cover** (or **minimal cover**) is a simplified set of **functional dependencies** (FDs) that is **equivalent** to the original set but has no extraneous attributes and no redundant FDs. It is used in database normalization to minimize redundancy and ensure that the database is efficient.

Steps to Find a Canonical Cover

1. **Decompose FDs:** Break down each FD with multiple attributes on the right-hand side into multiple FDs, each with a single attribute on the right.
2. **Remove Extraneous Attributes:** Check if any attribute in the left-hand side of an FD is unnecessary by testing whether the FD still holds after removing it.
3. **Remove Redundant FDs:** Check if any FD can be removed without affecting the equivalence of the set.

Example

Given Functional Dependencies:

```
plaintext
```

$$F = \{ A \rightarrow BC, B \rightarrow C, A \rightarrow B, AB \rightarrow C \}$$

Step 1: Decompose FDs

Break down FDs with multiple attributes on the right:

```
plaintext
```

$$F = \{ A \rightarrow B, A \rightarrow C, B \rightarrow C, AB \rightarrow C \}$$

Step 2: Remove Extraneous Attributes

1. Check $AB \rightarrow C$:

- Test if B is extraneous by computing the closure of A :
 - Closure of A : $\{A \rightarrow B \rightarrow C\}$ (using $A \rightarrow B$ and $B \rightarrow C$).
 - Hence, B is extraneous, and $AB \rightarrow C$ can be reduced to $A \rightarrow C$.

Updated FDs:

```
plaintext
```

$$F = \{ A \rightarrow B, A \rightarrow C, B \rightarrow C \}$$

2. No further extraneous attributes are present.

Step 3: Remove Redundant FDs

1. Check $B \rightarrow C$:

- Remove $B \rightarrow C$ and test equivalence of the set:
 - Remaining FDs: $\{A \rightarrow B, A \rightarrow C\}$
 - Closure of A : $\{A \rightarrow B \rightarrow C\}$ (using $A \rightarrow B$ and $A \rightarrow C$).
 - $B \rightarrow C$ is redundant.

Updated FDs:

```
plaintext
```

```
F = { A → B, A → C }
```

Canonical Cover

The canonical cover is:

```
plaintext
```

```
{ A → B, A → C }
```

Explanation of the Result

- The original set of FDs $\{ A \rightarrow BC, B \rightarrow C, A \rightarrow B, AB \rightarrow C \}$ is **equivalent** to $\{ A \rightarrow B, A \rightarrow C \}$.
- All extraneous attributes and redundant FDs are removed, making the set minimal.

Canonical cover simplifies database schema design and helps in normalization by reducing complexity and redundancy.