Normalization in DBMS with examples

Anomalies in DBMS

There are three types of anomalies that occur when the database is not normalized. These are – Insertion, update and deletion anomaly. Let's take an example to understand this.

Example: Suppose a manufacturing company stores the employee details in a table named employee that has four attributes: emp_id for storing employee's id, emp_name for storing employee's name, emp_address for storing employee's address and emp_dept for storing the department details in which the employee works. At some point of time the table looks like this:

emp_id	emp_name	emp_address	emp_dept
101	Rick	Delhi	D001
101	Rick	Delhi	D002
123	Maggie	Agra	D890
166	Glenn	Chennai	D900
166	Glenn	Chennai	D004

Normalization in DBMS with examples

The above table is not normalized. We will see the problems that we face when a table is not normalized.

Update anomaly: In the above table we have two rows for employee Rick as he belongs to two departments of the company. If we want to update the address of Rick then we have to update the same in two rows or the data will become inconsistent. If somehow, the correct address gets updated in one department but not in other then as per the database, Rick would be having two different addresses, which is not correct and would lead to inconsistent data.

Insert anomaly: Suppose a new employee joins the company, who is under training and currently not assigned to any department then we would not be able to insert the data into the table if emp_dept field doesn't allow nulls.

Delete anomaly: Suppose, if at a point of time the company closes the department D890 then deleting the rows that are having emp_dept as D890 would also delete the information of employee Maggie since she is assigned only to this department.

- **Normalization**
- Here are the most commonly used normal forms:
- First normal form(1NF)
- Second normal form(2NF)
- Third normal form(3NF)
- Boyce & Codd normal form (BCNF)

First normal form (1NF)

- As per the rule of first normal form, an attribute (column) of a table cannot hold multiple values. It should hold only atomic values.
- **Example**: Suppose a company wants to store the names and contact details of its employees. It creates a table that looks like this:

emp_id	emp_name	emp_address	emp_mobile
101	Herschel	New Delhi	8912312390
102	Jon	Kanpur	8812121212 9900012222
103	Ron	Chennai	7778881212
104	Lester	Bangalore	9990000123 8123450987

- Two employees (Jon & Lester) are having two mobile numbers so the company stored them in the same field as you can see in the table above.
- This table is **not in 1NF** as the rule says “each attribute of a table must have atomic (single) values”, the emp_mobile values for employees Jon & Lester violates that rule.
- To make the table complies with 1NF we should have the data like this:

emp_id	emp_name	emp_address	emp_mobile
101	Herschel	New Delhi	8912312390
102	Jon	Kanpur	8812121212
102	Jon	Kanpur	9900012222
103	Ron	Chennai	7778881212
104	Lester	Bangalore	9990000123
104	Lester	Bangalore	8123450987

Second normal form (2NF)

- A table is said to be in 2NF if both the following conditions hold:
- Table is in 1NF (First normal form)
- No non-prime attribute is dependent on the proper subset of any candidate key of table.
- An attribute that is not part of any candidate key is known as non-prime attribute.
- **Example**: Suppose a school wants to store the data of teachers and the subjects they teach. They create a table that looks like this: Since a teacher can teach more than one subjects, the table can have multiple rows for a same teacher.

teacher_id	subject	teacher_age
111	Maths	38
111	Physics	38
222	Biology	38
333	Physics	40
333	Chemistry	40

Candidate Keys: {teacher_id, subject}
Non prime attribute: teacher_age

- The table is in 1 NF because each attribute has atomic values. However, it is not in 2NF because non prime attribute teacher_age is dependent on teacher_id alone which is a proper subset of candidate key. This violates the rule for 2NF as the rule says “**no** non-prime attribute is dependent on the proper subset of any candidate key of the table”.
- To make the table complies with 2NF we can break it in two tables like this:

- teacher_details table

teacher_id	teacher_age
111	38
222	38
333	40

- teacher_subjects table

teacher_id	subject
111	Maths
111	Physics
222	Biology
333	Physics
333	Chemistry

- **Third Normal form (3NF)**
- A table design is said to be in 3NF if both the following conditions hold:
- Table must be in 2NF
- **Transitive functional dependency** of non-prime attribute on any super key should be removed.
- An attribute that is not part of any **candidate key** is known as non-prime attribute.

In other words 3NF can be explained like this: A table is in 3NF if it is in 2NF and for each functional dependency X-> Y at least one of the following conditions hold:

- X is a **super key** of table
- Y is a prime attribute of table
- An attribute that is a part of one of the candidate keys is known as prime attribute.
- **Example**: Suppose a company wants to store the complete address of each employee, they create a table named employee_details that looks like this:

emp_id	emp_name	emp_zip	emp_state	emp_city	emp_district
1001	John	282005	UP	Agra	Dayal Bagh
1002	Ajeet	222008	TN	Chennai	M-City
1006	Lora	282007	TN	Chennai	Urrapakkam
1101	Lilly	292008	UK	Pauri	Bhagwan
1201	Steve	222999	MP	Gwalior	Ratan

- **Super keys**: {emp_id}, {emp_id, emp_name}, {emp_id, emp_name, emp_zip}…so on
  Candidate Keys: {emp_id}
  Non-prime attributes: all attributes except emp_id are non-prime as they are not part of any candidate keys.

- Here, emp_state, emp_city & emp_district dependent on emp_zip. And, emp_zip is dependent on emp_id that makes non-prime attributes (emp_state, emp_city & emp_district) transitively dependent on super key (emp_id). This violates the rule of 3NF.
- To make this table complies with 3NF we have to break the table into two tables to remove the transitive dependency:

- **employee table:**
-

emp_id	emp_name	emp_zip
1001	John	282005
1002	Ajeet	222008
1006	Lora	282007
1101	Lilly	292008
1201	Steve	222999

- **employee_zip table:**

emp_zip	emp_state	emp_city	emp_district
282005	UP	Agra	Dayal Bagh
222008	TN	Chennai	M-City
282007	TN	Chennai	Urrapakkam
292008	UK	Pauri	Bhagwan
222999	MP	Gwalior	Ratan

Boyce Codd normal form (BCNF)

- It is an advance version of 3NF that's why it is also referred as 3.5NF. BCNF is stricter than 3NF. A table complies with BCNF if it is in 3NF and for every **functional dependency** X->Y, X should be the super key of the table.
- **Example**: Suppose there is a company wherein employees work in **more than one department**. They store the data like this:

emp_id	emp_nationality	emp_dept	dept_type	dept_no_of_emp
1001	Austrian	Production and planning	D001	200
1001	Austrian	stores	D001	250
1002	American	design and technical support	D134	100
1002	American	Purchasing department	D134	600

- **Functional dependencies in the table above**:
  emp_id -> emp_nationality
  emp_dept -> {dept_type, dept_no_of_emp}
- **Candidate key**: {emp_id, emp_dept}
- The table is not in BCNF as neither emp_id nor emp_dept alone are keys.
- To make the table comply with BCNF we can break the table in three tables like this:

- **emp_nationality table:**

emp_id	emp_nationality
1001	Austrian
1002	American

- **emp_dept table:**

emp_dept	dept_type	dept_no_of_emp
Production and planning	D001	200
stores	D001	250
design and technical support	D134	100
Purchasing department	D134	600

- **emp_dept_mapping table:**

emp_id	emp_dept
1001	Production and planning
1001	stores
1002	design and technical support
1002	Purchasing department

- **Functional dependencies**:
  emp_id -> emp_nationality
  emp_dept -> {dept_type, dept_no_of_emp}
- **Candidate keys**:
  For first table: emp_id
  For second table: emp_dept
  For third table: {emp_id, emp_dept}

This is now in BCNF as in both the functional dependencies left side part is a key.