Anomalies in DBMS

Problems that can occur in poorly planned, unnormalized databases where all the data is stored in one table (a flat-file database).

Table: University

Sid	Sname	Cid	Cname	Fid	Fname	Salary
1	Ram	C1	DBMS	F1	Sachin	30000
2	Shyam	C2	Java	F2	Boby	28000
3	Ankit	C1	DBMS	F1	Sachin	30000
4	saurabh	C1	DBMS	F1	Sachin	30000

- **Insert Anomaly:** An Insert Anomaly occurs when certain attributes cannot be inserted into the database without the presence of other attributes.

Table: University

Sid	Sname	Cid	Cname	Fid	Fname	Salary
1	Ram	C1	DBMS	F1	Sachin	30000
2	Shyam	C2	Java	F2	Boby	28000
3	Ankit	C1	DBMS	F1	Sachin	30000
4	saurabh	C1	DBMS	F1	Sachin	30000
				F3	Arun	29000

Insertion Anomaly

- Suppose a new faculty joins the University, and the Database Administrator inserts the faculty data into the above table. But he is not able to insert because Sid is a primary key, and can't be NULL. So this type of anomaly is known as an insertion anomaly.
- **Delete Anomaly:** A Delete Anomaly exists when certain attributes are lost because of the deletion of other attributes.

When the Database Administrator wants to delete the student details of Sid=2 from the above table, then it will delete the faculty and course information too, which cannot be recovered further.

SQL:

Sid	Sname	Cid	Cname	Fid	Fname	Salary
1	Ram	C1	DBMS	F1	Sachin	30000
~~2~~	~~Shyam~~	Deletion anomaly				
3	Ankit	C1	DBMS	F1	Sachin	30000
4	Saurabh	C1	DBMS	F1	Sachin	30000

DELETE FROM *University* WHERE *Sid=2*;

- **Update Anomaly:** An Update Anomaly exists when one or more instances of duplicated data are updated, but not all

  When the Database Administrator wants to change the salary of faculty F1 from 30000 to 40000 in the above table University, then the database will update the salary in more than one row due to data redundancy. So, this is an update anomaly in a table.

SQL:

UPDATE *University* SET
***Salary= 40000* WHERE**
***Fid="F1"*;**

Sid	Sname	Cid	Cname	Fid	Fname	Salary
1	Ram	C1	DBMS	F1	Sachin	30000
2	Shyam	C2	Java	F2	Boby	28000
3	Ankit	C1	DBMS	F1	Sachin	30000
4	saurabh	C1	DBMS	F1	Sachin	30000

Database Normalization Form Cheat Sheet

Database normalization removes redundancy and ensures that data is logically stored. It removes insertion, deletion, and updation anomalies. Here, I'll briefly explain the 5 normalization rules.

1. First Normal Form
2. Second Normal Form
3. Third Normal Form
4. BCNF: Boyce and Codd Normal Form
5. Fourth Normal Form

First Normal Form

A table is supposed to be in first normal form if,

- All the attributes are single-valued (atomic).
- All the columns have unique names.
- The order in which data is stored does not matter.

Example: In the Shirt_Info table, the Size attribute is not atomic. Hence, it can be decomposed

Shirt_Info		
Shirt_ID	Design	Size
100	Mickey Mouse	XXL,XL,M
101	Flowers	M
102	Good Vibes	M,S
103	Triangles	XL
Primary Key: Shirt_ID		

Design_Info	
Shirt_ID	Design
100	Mickey Mouse
101	Flowers
102	Good Vibes
103	Triangles
Primary Key: Shirt_ID	

Size_Info	
Shirt_ID	Size
100	XXL
100	XL
100	M
101	M
102	M
102	S
103	XL
Primary Key: Shirt_ID, Size	

into Design_Info and Size_Info as shown in the image.

Second Normal Form

A table is supposed to be in second normal form if,

- It is in the 1st normal form.
- It does not have any partial dependency

Example: In the Customer_Info table, Store_Name depends on Store_ID and not on Cust_ID. This is a partial dependency. Hence, Customer_Info is not in second normal form (though it satisfies 1NF). It can be decomposed into Customer_Data and Store_Data as shown below.

Customer_Info			
Cust_ID	Store_ID	Customer_Name	Store_Name
568	74896	Liam	JPM_1
574	74896	Charles	JPM_1
568	25614	Liam	JPM_2
235	25614	Cyrus	JPM_2
Primary Key: Cust_ID, Store_ID			

Customer_Data	
Cust_ID	Customer Name
568	Liam
574	Charles
145	Marcus
235	Cyrus
Primary Key: Cust_ID	

Store_Data	
Store_ID	Store_Name
74896	JPM_1
25614	JPM_2
Primary Key: Store_ID	

Third Normal Form

A table is supposed to be in third normal form if,

- It satisfies 2nd normal form.
- It does not have any transitive dependencies.

Example: Movie_Info is in second normal form, but it has a transitive dependency. Therefore, it is not in third normal form. It can be decomposed into Movie_Rating and Genre_Info as shown below.

Movie_Info			
MID	GENRE_ID	GENRE	RATING
1	G1	Comedy	4.2
2	G2	Romance	4.5
3	G3	Action	2.1
4	G4	Horror	3.5
Primary Key: MID			

Movie_Rating		
MID	RATING	GENRE_ID
1	4.2	G1
2	4.5	G2
3	2.1	G3
4	3.5	G4
Primary Key: MID		

Genre_Info	
GENRE_ID	GENRE
G1	Comedy
G2	Romance
G3	Action
G4	Horror
Primary Key: GENRE_ID	

In Movie_Info, MID -> GENRE_ID -> GENRE
Which implies MID -> GENRE. Hence, there is a transitive dependency ie. It is not in 3rd NF.

BoYCE-Codd Normal Form (BCNF)

A table is supposed to be in BCNF if,

- It is in 3rd Normal Form.
- FOr every dependency X->Y, X cannot be a non-prime attribute if Y is a prime attribute (i.e., X should be a super key)

Example: Student_Info is not in BCNF because in the dependency Faculty -> Subject, Subject is a prime attribute and Faculty is a non-prime attribute. It can be decomposed into Student_Faculty_Info and Faculty_Info as shown below.

Also read about attribute closure, testing for BCNF, and the BCNF Decomposition Algorithm.

Student_Info		
Student_ID	Subject	Faculty
101	DBMS	Dan
101	Data Science	Blair
102	DBMS	Cyrus
103	Cloud Computing	Serena
104	DBMS	Dan
Primary Key: Student_ID, Subject		

Student_Faculty_Info	
SID	FID
101	1
101	2
102	3
103	4
104	1
Primary Key: SID, FID	

Faculty_Info		
FID	Faculty	Subject
1	Dan	DBMS
2	Blair	Data Science
3	Cyrus	DBMS
4	Serena	Cloud Computing
Primary Key: FID		

(Assume that a faculty takes only one course.)
In Student_Info, (Student_ID, Subject) -> Faculty
But, Faculty -> Subject where Faculty is non-prime and Subject is prime attribute.
Hence it is not in BCNF.

Fourth Normal Form
A table is supposed to be in fourth normal form if,

- It is in BCNF.
- It has no multi-valued dependency.

Multi-valued Dependency: *For a dependency A → B, if for a single value of A, multiple values of*

Student_Info		
SID	Course	Hobby
1	Computer	Writing
1	Math	Coding
2	Chemistry	Writing
3	Biology	Cricket
4	Physics	Cooking

Student_Course	
SID	Course
1	Computer
1	Math
2	Chemistry
3	Biology
4	Physics
Primary Key: SID, Course	

Student_Hobby	
SID	Hobby
1	Writing
1	Coding
2	Writing
3	Cricket
4	Cooking
Primary Key: SID, Hobby	

In student_info, SID=1 gives more than one record. Hence, it is a multi-valued dependency.

B exist, then the relation will be a multi-valued dependency.

Student ID	Semester	Lecture	TA
1234	6	Numerical Methods	John
1221	4	Numerical Methods	Smith
1234	6	Visual Computing	Bob
1201	2	Numerical Methods	Peter
1201	2	Physics II	Simon

Functional Dependency

- Whenever two rows in this table feature the same StudentID, they also necessarily have the same Semester values. This basic fact can be expressed by a functional dependency:

StudentID → Semester.

Transitive Dependency

- A functional dependency is said to be transitive if it is indirectly formed by two functional dependencies. For e.g.
- X -> Z is a transitive dependency if the following three functional dependencies hold true:

X- -> >Y

Y does not ->X

Y->Z

Let's take an example to understand it better:

Book	Author	Author_age
Windhaven	George R. R. Martin	66
Harry Potter	J. K. Rowling	49
Dying of the Light	George R. R. Martin	66

{Book} ->{Author} (if we know the book, we know the author's name)

{Author} does not ->{Book

{Author} -> {Author_age}

Therefore, as per the rule of **transitive dependency**, {Book} -> {Author_age} should hold, which makes sense because if we know the book name, we can know the author's age.

Partial Dependency

- If a non-prime attribute can be determined by the part of the candidate key in a relation, it

Example of partial Dependency: Suppose there is a relation **R** with attributes **A, B,** and **C.**

R(A,B,C)

Where,

{AB} is a candidate key.

{C} is a non-prime attribute.

Then,

{A, B} are the prime attributes.

A → C is a partial dependency.

Part of a candidate key **Non- prime attribute**

is known as a partial dependency.

Example to convert a given 1NF to 3NF

Imagine we're building a restaurant management application. That application needs to store data about the company's employees, and it starts by creating the following table of employees:

employee_id	name	job_code	job	state_code	home_state
E001	Alice	J01	Chef	26	Michigan
E001	Alice	J02	Waiter	26	Michigan
E002	Bob	J02	Waiter	56	Wyoming
E002	Bob	J03	Bartender	56	Wyoming
E003	Alice	J01	Chef	56	Wyoming

All the entries are atomic, and there is a composite primary key (employee_id, job_code), so the table is in the **first normal form (1NF)**.

But even if you only know someone's employee_id, then you can determine

their name, home_state, and state_code (because they should be the same person). Thismeans name, home_state, and state_code are dependent on employee_id (a part of the primary composite key). So, the table is not in **2NF**. We should separate them into a different table to make it 2NF.

Example of Second Normal Form (2NF)

employee_roles Table

employee_id	job_code
E001	J01
E001	J02
E002	J02
E002	J03
E003	J01

employees Table

employee_id	name	state_code	home_state
E001	Alice	26	Michigan
E002	Bob	56	Wyoming
E003	Alice	56	Wyoming

jobs table

job_code	job
J01	Chef
J02	Waiter
J03	Bartender

The home state is now dependent on state_code. So, if you know the state_code, then you can find the home_state value.

To take this a step further, we should separate them again into a different table to make it 3NF.

Example of Third Normal Form (3NF)

employee_roles Table

employee_id	**job_code**
E001	J01
E001	J02
E002	J02
E002	J03
E003	J01

employees Table

employee_id	**name**	**state_code**
E001	Alice	26
E002	Bob	56
E003	Alice	56

jobs Table

job_code	**job**
J01	Chef
J02	Waiter
J03	Bartender

states Table

state_code	**home_state**
26	Michigan
56	Wyoming

Now our database is in 3NF.

Example

Consider the following Relation, check the highest normal form (i.e., up to BCNF)? Assume it is in 1NF.

R(ABCDEF)

FD are {AB□C, C□DE, E□F, F□A}

Solution

Step 1: Find all the candidate keys using the closure method

AB+=ABCDEF

(Candidate key should be minimal, here AB is known as a candidate key or super key)

Check A+=A

B+=B not a C.K. because CK should determine all the attributes of relation R, so A & B alone are not CK, but AB is CK

To find other CK, check whether either A or B is present at the RHS of given FDs; if yes, then replace it & check for CK again.

i.e, for F□A

For AB, replace A by F.

i.e. FB+=FBACDE

Similarly, for E□F

EB+=EBFACD

For C□DE

CB+=CBDEFA

For AB□C

AB is already a CK

So finally we found CK as {AB, FB, EB, CB}

STEP 2 Write all prime attributes, i.e, attributes that participate in making CK

i.e {A, B, C, E, F}

STEP 3 Write all Nonprime attributes, i.e, {D

Rule for

BCNF – LHS of all the FD's should be Candidate Key or Super Key

3rd NF – Relation should not be in Transitive Dependence

OR

LHS should be CK, **OR** RSH should be prime attribute

2nd NF: should not be

LHS should be a proper subset of CK, **AND** RSH should be Non non-prime attribute

	AB□C	C□DE	E□F	F□A	RESULT
BCNF	✓	X	X	X	NOT IN BCNF
3RD NF	✓	X	✓	✓	NOT IN 3RD NF
2ND NF	✓	X	✓	✓	NOT IN 2ND NF
1ST NF	✓	✓	✓	✓	IN 1ST NF

Note: Even a single cross can tell not in that NF