CHAPTER – 11

NORMALIZATION

Prepared By: Prof. Kinjal Acharya
Assistant Professor
DDU, Nadiad

INTRODUCTION

- Normalization is a database design technique which organizes tables in a manner that reduces redundancy and dependency of data.
- It divides larger tables to smaller tables and link them using relationships.
- Normalization is a systematic approach of decomposing tables to **eliminate data redundancy and undesirable characteristics like Insertion, Update and Deletion Anomalies.**
- There are three types of anomalies that occur when the database is not normalized: Insert, Update and Delete.
- Example: Suppose a manufacturing company stores the employee details in a table named employee that has four attributes: emp_id, emp_name, emp_address and emp_dept. 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

ANOMALIES IN DATABASE

- **Update anomaly:** In the emp table there are 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.
- To overcome these anomalies we need to normalize the data.

NONLOSS DECOMPOSITION AND FUNCTIONAL DEPENDENCIES

- Normalization procedure involves breaking down or decomposing a given relvar into other relvars, and moreover that the decomposition is required to be reversible, so that no information is lost in the process; in other words, the only decomposition we are interested in are ones that are indeed nonloss.
- **Example**, consider the suppliers relvar S, with heading { S#, STATUS, CITY }.
- Given figure shows a sample value for this relvar and in the parts of the figure labelled (a) and (b) – two possible decompositions corresponding to that sample value.

S

S#	STATUS	CITY
S3	30	Paris
S5	30	Athens

(a) SST

S#	STATUS
S3	30
S5	30

SC

S#	CITY
S3	Paris
S5	Athens

(b) SST

S#	STATUS
S3	30
S5	30

STC

STATUS	CITY
30	Paris
30	Athens

Examining the two decompositions, we observe that :

- In case (a), no information is lost; the SST and SC value still tell us that supplier S3 has status 30 and city Paris, and supplier S5 has status 30 and city Athens. In other words, this first decomposition is indeed **nonloss**.
- In case (b), by contrast, information definitely is lost; we can still tell that both suppliers have status 30, but we cannot tell which supplier has which city. In other words, the second decomposition is not nonloss but **lossy**.

NONLOSS DECOMPOSITION AND FUNCTIONAL DEPENDENCIES

- Observe next that when we say in Case (a) that no information is lost, what we really mean is that if we, join SST and SC back together again, we get back to the original S.
- Observe that the process of decomposition is really a process of projection; SST, SC, and STC in the figure are each projections of the original relvar S. So the **decomposition operator in the normalization procedure is in fact projection.**
- Observe next that in Case(a) no information is lost means if we join SST and SC back together again, we get back the original S.
- In Case (b), by contrast, if we join SST and SC together again, we do not get back the original S, and so we have lost information.
- In other words, **reversibility means that the original relvar is equal to the join of its projections.**
- The **decomposition operator for normalization purposes is projection, so the recomposition operator is join.**
- If R1 and R2 are projections of some relvar R, and R1 and R2 between them include all of the attributes of R, what conditions have to be satisfied in order to guarantee that joining R1 and R2 back together takes us back to the original R? And this is where functional dependencies come in.
- Returning to our example, observe that relvar S satisfies the irreducible set of FDs.

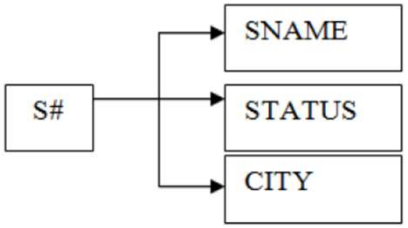
  S# → STATUS

  S# → CITY

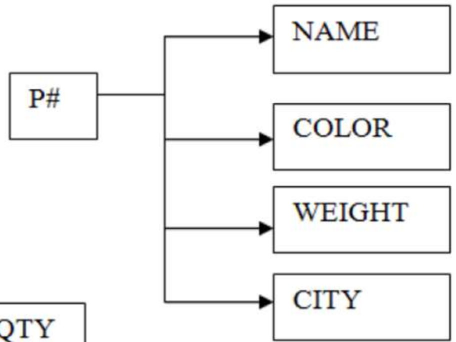
- Given the fact that it satisfies these FDs, relvar S is equal to the join of its projections on {S#, STATUS} and {S#, CITY}.

HEATH'S THEOREM

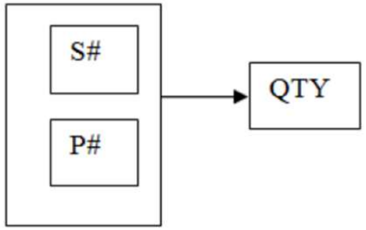
- Let R {A,B,C} be a relvar, where A, B, and C are sets of attributes. If R satisfies the FD A □ B, then R is equal to the join of its projections on {A, B} and { A,C}.
- Taking A as S#, B as STATUS and C as CITY, this theorem confirms that relvar S can be nonloss decomposed into its projections on, {S#, STATUS} and { S#, CITY }.
- **Left-irreducible FDs :** FD is said to be left-irreducible if its left-hand side cannot be further reduced without loss of information. For example, relvar SCP satisfies the FD { S#, P# } → CITY
- Attribute P# on left-hand side here is redundant for FD purposes; that is, we also have the FD S# →CITY (i.e., CITY is functionally dependent on S# alone). This latter FD is left-irreducible, but the previous one is not.
- **FD diagrams:** Let R be a relvar and let I be some irreducible set of FDs that apply to R. It is convenient to represent the set I by means of a functional dependency diagram (FD diagram).



FD Diagram for S



FD Diagram for P



FD Diagram for SP

NORMAL FORMS

- Here are the most commonly used normal forms:

1. First normal form(1NF)
2. Second normal form(2NF)
3. Third normal form(3NF)
4. Boyce &Codd normal form (BCNF)

- A relvar is said to be in a particular normal form if it satisfies a certain prescribed set of conditions.
- By **Normalization procedure**, a relvar that is in some given normal form, say 2NF, can be replaced by a set of relvars in some more desirable form, say 3NF.
- Normalization procedure is **successive reduction of a given collection of relvars to some more desirable form.**
- The procedure is **reversible.** It is always possible to take the output from the procedure (say the set of 3NF relvars) and map it back to the input (say the original 2NF relvar).
- Reversibility is important because it means the normalization process is **information – preserving**.

FIRST NORMAL FORM(1NF)

- A relvar is in 1NF if and only if, **in every legal value of that relvar, every tuple contains exactly one value for each attribute.**
- As per the rule of first normal form, an attribute 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.

FIRST NORMAL FORM(1NF)

- 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

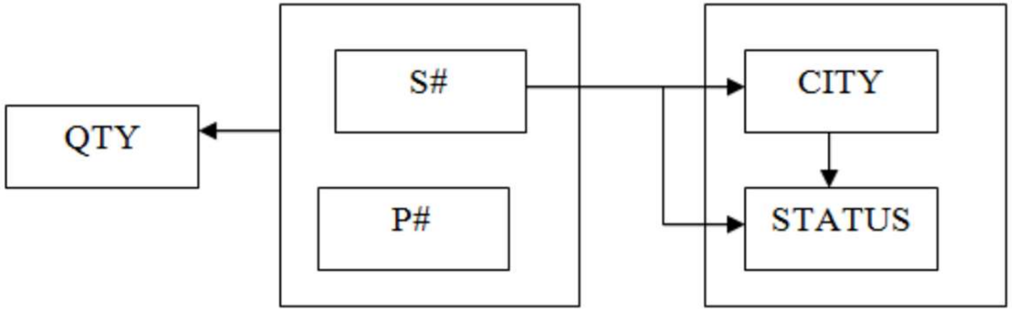
- Let us suppose that information concerning suppliers and shipments, rather than being split into the two relvars S and SP, is lumped together into a single relvar as follows:

FIRST { S#, STATUS, CITY, P#, QTY } PRIMARY KEY { S#, P# }

- This relvar is an extended version of SCP. The attributes have their usual meanings, except that for the sake of the example we introduce an additional constraint: CITY → STATUS
- (STATUS is functionally dependent on CITY ; the meaning of this constraint is that a supplier's status is determined by the location of that supplier - e.g., all London suppliers must have a status of 20).

FIRST NORMAL FORM(1NF)

- FD diagram for FIRST:



S#	STATUS	CITY	P#	QTY
S1	20	London	P1	300
S1	20	London	P2	200
S1	20	London	P3	400
S1	20	London	P4	200
S1	20	London	P5	100
S1	20	London	P6	100
S2	10	Paris	P1	300
S2	10	Paris	P2	400
S3	10	Paris	P2	200
S4	20	London	P2	200
S4	20	London	P4	300
S4	20	London	P5	400

1NF: ANOMALIES

- The redundancies in relvar FIRST lead to insert, update, delete anomalies.

1. **INSERT :** We cannot insert the fact that a particular supplier is located in a particular city until that supplier supplies at least one part.
2. **DELETE :** If we delete the sole FIRST tuple for a particular supplier, we delete not only the shipment connecting that supplier to a particular part but also the information that the supplier is located in particular city. For example, if we delete the FIRST tuple with S# value S3 and P# value P2, we lose the information that S3 is located in Paris.
3. **UPDATE :** The city value for a given supplier appears in FIRST many times, in general. This redundancy causes update problems. For example, if supplier S1 moves from London to Amsterdam, we are faced with either the problem of searching FIRST to finding every tuple connecting S1 and London (and changing it) or the possibility of producing an inconsistent result.

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 primary key of table.An attribute that is not part of any primary key is known as non-prime attribute.
- A relvar is in 2NF if and only if it is in 1NF and every nonkey attribute is irreducibly dependent on the primary key.
- Lets apply 2NF on FIRST relvar:
- In FIRST attributes STATUS and CITY are dependent on S# only which is a proper subset of primary key {S#, P#} which violates the 2NF.
- So we can normalize FIRST in two relvars SECOND1 and SECOND2 as follow:

  SECOND1 {S#, STATUS, CITY} PRIMARY KEY {S#}

  SECOND2 {S#, P#, Qty} PRIMARY KEY {S#, P#}
- This decomposition is not lossy as it still preserves all the FDs available on FIRST.
- Check for anomalies on SECOND1 and SECOND2 on your own.
- Consider another 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.

SECOND NORMAL FORM(2NF)

Primary Key: {teacher_id, subject}
Non prime attribute: teacher_city

Teacher_ID	Subject	Teacher_City
111	Maths	Nadiad
111	Physics	Nadiad
222	Biology	Anand
333	Physics	Vadodara
333	Chemistry	Vadodara

The table is in 1 NF because each attribute has atomic values.
• But it is not in 2NF because non prime attribute teacher_City is dependent on teacher_id alone which is a proper subset of primary key. • This violates the rule for 2NF.
To make the table complies with 2NF we can break it in two tables like this:

teacher_subject:

Teacher_ID	Teacher_City
111	Nadiad
222	Anand
333	Vadodara

teacher_details:

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:

1. **Table must be in 2NF**
2. **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.
- A relvar is in 3NF if and only if it is in 2NF and every nonkey attribute is non transitively dependent on the primary key.
- The second step in the normalization procedure is to take projections to eliminate transitive dependencies.
- Suppose given a relvar R as follows :

  R { A, B, C } **PRIMARY KEY { A }**

- **FDs:** A → B and B → C
- The normalization discipline recommends replacing R by its two projections R1 and R2, as follows:

1. R1 { B, C } PRIMARY KEY { B }
2. R2 { A, B } PRIMARY KEY { A } FOREIGN KEY { B } REFERENCES R1

- So we can say that SECOND1 is not in 3NF as there is a dependency between non prime attributes that is status is dependant on city.}
- THIRD1 {S#, City} PRIMARY KEY {S#} FOREIGN KEY {City} REFERENCES THIRD2
- THIRD2 {City, Status} PRIMARY KEY {City}

THIRD NORMAL FORM (3NF)

- 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	DayalBagh
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} … 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 {emp_id, emp_name, emp_zip} PRIMARY KEY {emp_id} FOREIGN KEY{emp_zip} REFERENCES emp_zip
- Emp_zip {emp_zip, emp_state, emp_city, emp_district} PRIMARY KEY {emp_zip}

BOYCE CODD NORMAL FORM (BCNF)

- It is an advance version of 3NF that's why it is also referred to 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.

TEACH

STUDENT	COURSE	INSTRUCTOR
Narayan	Database	Mark
Smith	Database	Navathe
Smith	Operating Systems	Ammar
Smith	Theory	Schulman
Wallace	Database	Mark
Wallace	Operating Systems	Ahamad
Wong	Database	Omiecinski
Zelaya	Database	Navathe

A relation TEACH that is in 3NF but not BCNF.

- FDl: {STUDENT, COURSE} → INSTRUCTOR
- FD2: INSTRUCTOR → COURSE

BOYCE CODD NORMAL FORM (BCNF)

- Another example: Suppose there is a company wherein employees work in **more than one department**. They store the data like this: 3

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 and 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

BOYCE CODD NORMAL FORM (BCNF)

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

BOYCE CODD NORMAL FORM (BCNF)

- **Functional dependencies**:

  emp_id → emp_nationality

  emp_dept → {dept_type, dept_no_of_emp}

- **Candidate keys**:

1. For first table: emp_id
2. For second table: emp_dept
3. For third table: {emp_id, emp_dept}

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

DEPENDENCY PRESERVATION

- Suppose we are given some relvar R, which – after we have applied all steps of the normalization procedure – we replace R1, R2, ….Rn (all of them projections of R, of course).
- Let the set of given FDs, for the original relvar R be S, and let the sets of FDs that apply to relvars R1, Rs…. Rn be S1, S2… Sn respectively.
- Each FD in the set Si will refer to attributes of Ri only (i=1, 2,….,n). Enforcing the constraints (FDs) in any given set Si is thus a simple matter.
- But what we need to do is to enforce the constraints in the original set S.
- We would therefore like the decomposition into R1, Rs… Rn to be such that enforcing the constraints in S1, S2….., Sn individually is together equivalent to enforcing the constraints in the original set S – in other words, we would like the decomposition to be **dependency – preserving.**

DENORMALIZATION

- Let R1,………Rn be a set of relvars, then denormalizing those relvars means replacing them by their join R, such that for all possible values r1,….rn of R1,…..Rn projecting the corresponding value r of R over the attributes of Ri is guaranteed to yield ri again (i=1,….,n).
- Denormalization is necessary to achieve good performance.
- Full normalization means lots of logically separate relvars.
- Lots of logically separate relvars means lots of physically separate stored file;
- Lots of physically separate stored files means lots of I/O.
- The overall objective is to increase redundancy, by ensuring that R is at a lower level of normalization than the relvars R1,…Rn.
- The objective is to reduce the number of joins that need to be done at run time by doing some of those joins ahead of time, as a part of the database design.
- Denormalization that is done at the physical level only. When we say that denormalization is "goodfor performance," we really mean it is good for the performance of specific applications.

NORMALIZATION: EXAMPLE - 1

(b) DEPARTMENT

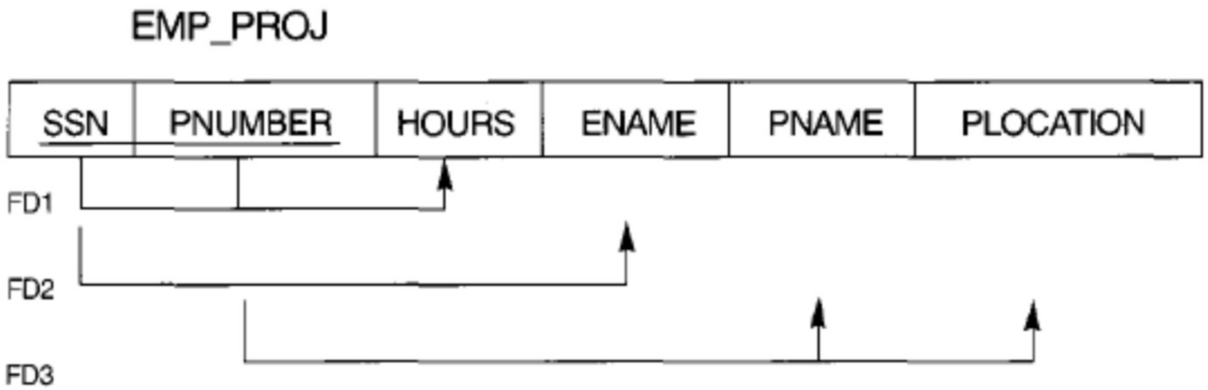
DNAME	DNUMBER	DMGRSSN	DLOCATIONS
Research	5	333445555	{Bellaire, Sugarland, Houston}
Administration	4	987654321	{Stafford}
Headquarters	1	888665555	{Houston}

(c) DEPARTMENT

DNAME	DNUMBER	DMGRSSN	DLOCATION
Research	5	333445555	Bellaire
Research	5	333445555	Sugarland
Research	5	333445555	Houston
Administration	4	987654321	Stafford
Headquarters	1	888665555	Houston

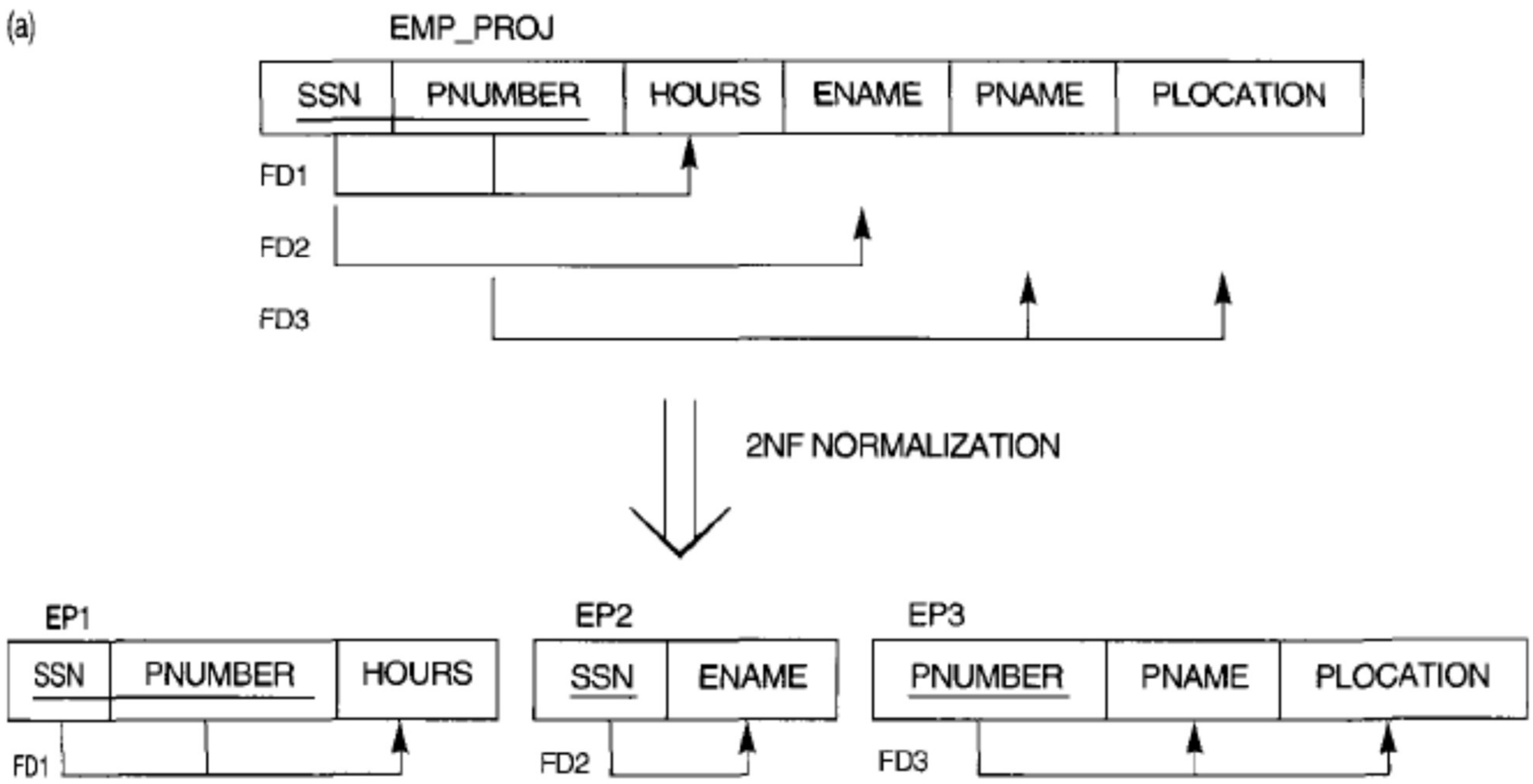
Normalization into 1NF. (a) A relation schema that is not in 1NF. (b) Example state of relation DEPARTMENT. (c) 1NF version of same relation with redundancy.

NORMALIZATION: EXAMPLE - 2



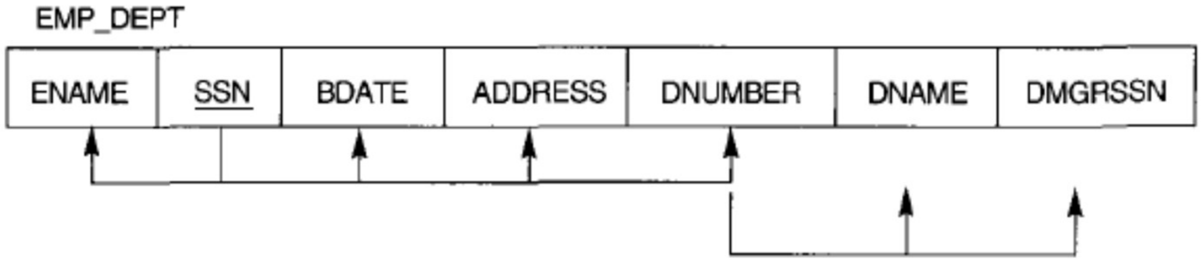
- The nonprime attribute ENAME violates 2NF because of FD2and nonprime attributes PNAME and PLOCATION because of FD3.
- The functional dependencies FD2 and FD3 make ENAME, PNAME, and PLOCATION partially dependent on the primary key {SSN, PNUMBER} of EMP_PROJ, thus violating the 2NF.

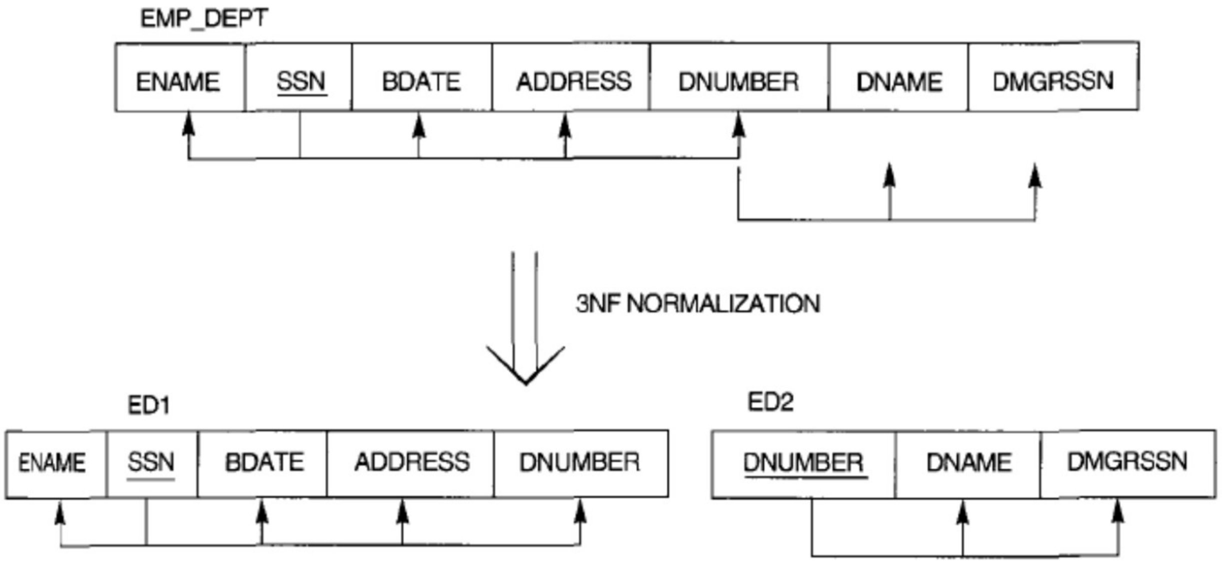
2NF OF EXAMPLE - 2



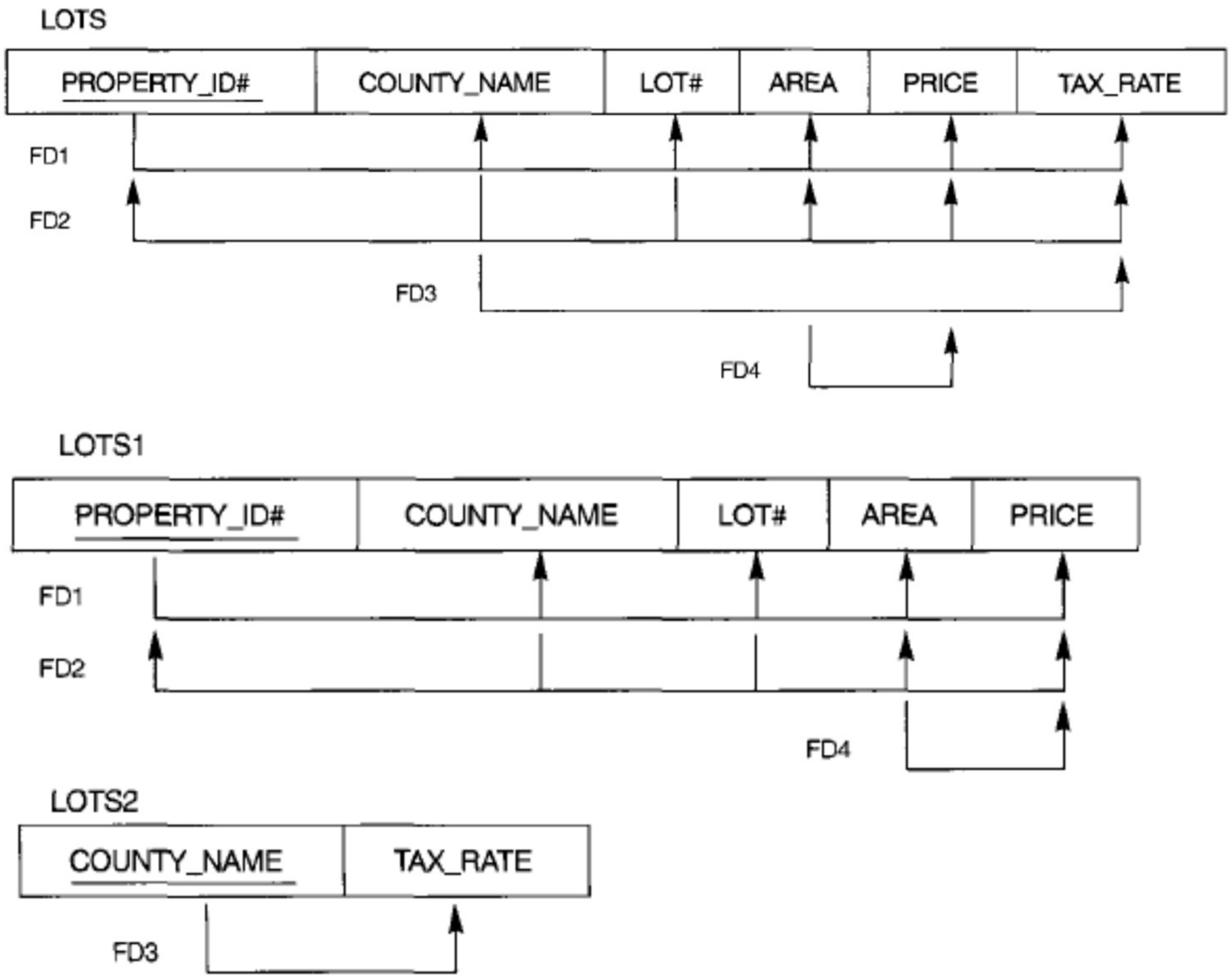
NORMALIZATION: EXAMPLE - 3

Emp_DEPT relation with FDs:
SSN → ENAME, BDATE, ADDRESS, DNUMBER
DNUMBER → DNAME, DMGRSSN

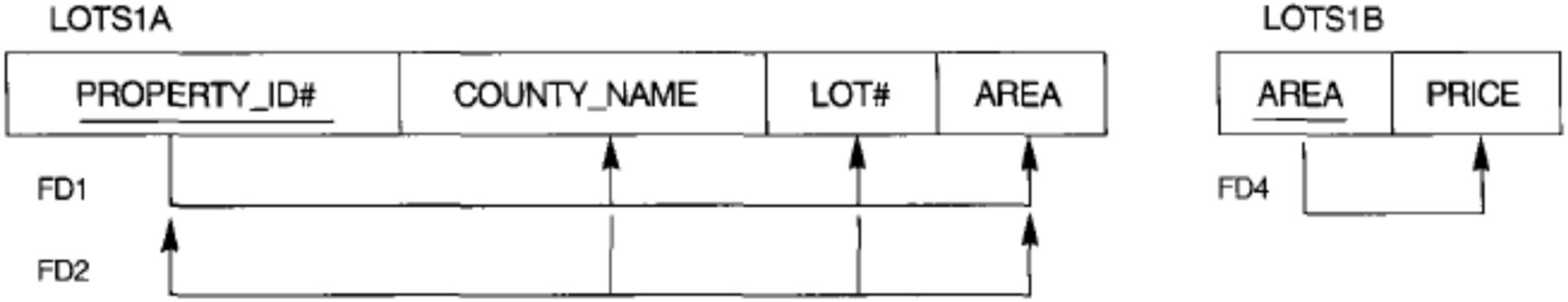


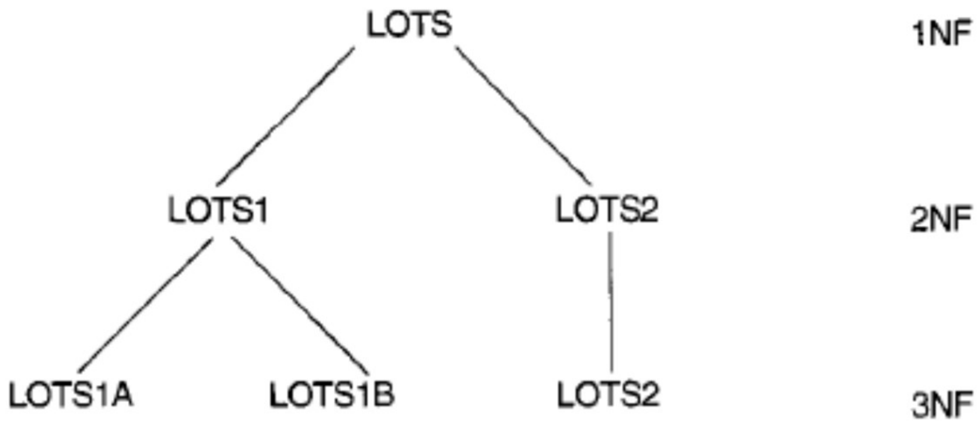


NORMALIZATION: EXAMPLE - 4



NORMALIZATION: EXAMPLE - 4





NORMALIZATION: EXAMPLE - 4

- If area is fixed for counties, one more FD is added.

  AREA→ COUNTY_NAME

- Here, AREA is not a super key. Hence, the table is not in BCNF.