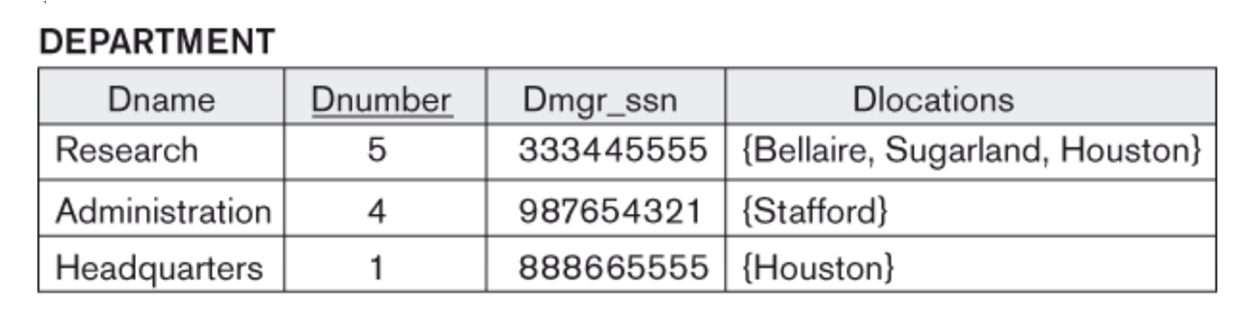
**Normalization and Functional Dependencies**

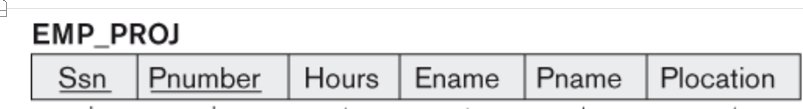
**What is the problem with the following:**

**a)**



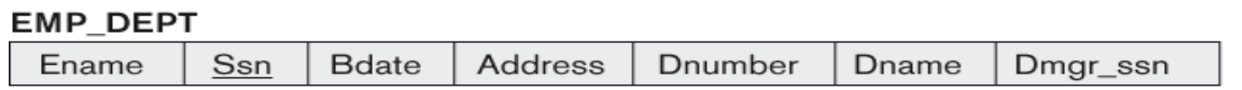
**Dlocations is not atomic**

b)



**Plocation and Pname is repeated for every employee who is working on the same project**

**c)**



**Dname and Dmgr ssn is repeated for everyone with the same department**

**Anomalies (Update, Delete, Insert)**

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



An Update Anomaly exists when one or more instances of duplicated data is updated, but not all.

A Delete Anomaly exists when certain attributes are lost because of the deletion of other attributes.



**id name address dept**

101 Rick CA D001

101 Rick CA D002

How do you fix it? 1st, 2nd, 3rd, BCNF normal form (It is a hierarchy)

• First Normal Form (1NF): Atomic and no repeating groups (no multivalued attributes)

• Second Normal Form (2NF): 1NF and no partial dependencies

• Third Normal Form (3NF) 2NF and no transitive dependencies.

• Boyce-Codd Normal Form (BCNF) 1NF and all dependencies from full key

**What are some of the concepts that we are going to discuss?**

* + - Functional dependency, F closure, X closure, Super key, Candidate key,
    - Armstrong axioms and inference rules, Bernstein synthesis, minimal cover,
    - Prime and non-prime attributes

**F closure {F}+**: The closure of F, denoted as F+, is the set of all regular FDs that can be derived from F

**X closure {X}+ :** The set of all those attributes which can be functionally determined from an attribute set is called as a closure of that attribute set.

**Superkey**: A superkey is a combination of columns that uniquely identifies any row within a RDBMS table.

**Candidate key**: A candidate key is a closely related concept where the superkey is reduced to the minimum number of columns required to uniquely identify each row.

**Minimal cover**:  Functional dependencies F covers another set of functional dependencies G, if every functional dependency in G can be inferred from F. F is a minimal cover of G if F is the smallest set of functional dependencies that cover G.

1. Right Hand Side (RHS) of all FDs should be single attribute.

2. Remove extraneous attributes.

3. Eliminate redundant functional dependencies.

**Prime/non-prime attributes**: Attributes that form a candidate key of a relation are called prime attributes. The rest of the attributes of the relation are non-prime.

**Armstrong’s axioms and inference rules:** Armstrong's axioms are a set of axioms used to infer all the functional dependencies on a relational database.  They are both sound and complete.

**Functional Dependency**

The functional dependency is a relationship that exists between two attributes. It typically exists between the primary key and non-key attribute within a table. The left side of FD is known as a determinant, the right side of the production is known as a dependent.

**SSN LNAME FNAME**

111 Jones Jack

111 Jones Jack

112 Jones jim

113 Jeff jim

**Name subject**

Joe prog

Joe prog

Joe econ

Jill econ

**Teacher Course Text**

Smith Data Structures Bartram

Smith Data Management Martin

Hall Data Structures Horowitz

Sly C++ Horowitz

**Dname Dnumber Dmgr\_ssn Dlocation**

Research 5 334 Bell

Research 5 334 Sugar

Research 5 334 Bell

Admin 4 335 Stafford

Headquarter 1 111 Sugar

**A B C**

1 2 3

1 2 3

2 3 4

2 4 4

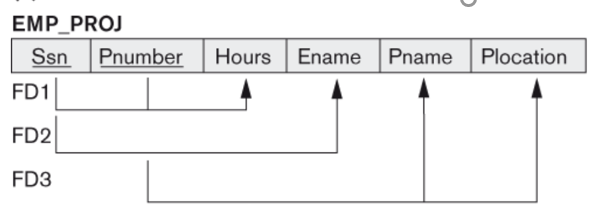
3 4 7

x->y means that y is functionally dependent on x or that x uniquely determines y.



ABC-> d, this means d is dependent fully on ABC and not a subset of the determinants

So AB->d would be wrong.

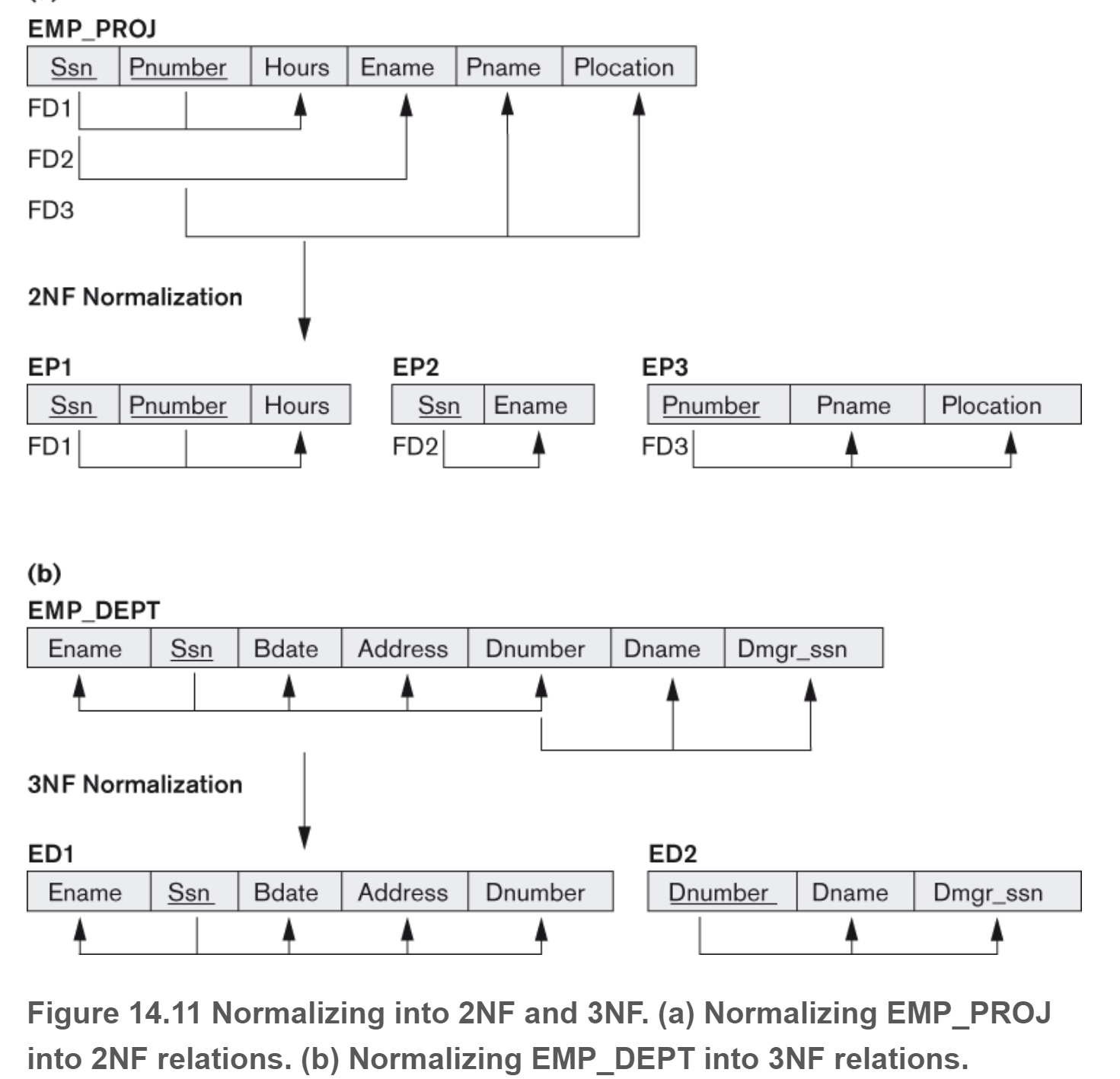


SSN, Pnumber-> Hours

SSN -> Ename

Pnumber -> Pname, Plocation

**Decomposition**



**Keys (Super key and candidate key)**

Emp\_SSN Emp\_Number Emp\_Name

--------- ---------- --------

1 226 jack

9 227 jill

3 228 jim

4 229 jim

Super key

Empssn, emp\_number, Emp\_name

Empssn, emp\_number

Emp\_ssn, emp\_name

Emp\_number, emp\_name

Emp\_ssn,

Emp\_number

Candidate key is because you can thin it out and you still have uniqueness

Emp\_snn

Emp\_number

**Applying Armstrong’s axioms and inference rules**



**Armstrong’s Axioms**

• Reflexivity: If Y ⊆ X , then X → Y

• Augmentation: If X → Y , then XZ → Y Z

• Transitivity: If X → Y and Y → Z, then X → Z

**Inference Rules**

• Union: If X → Y and X → Z, then X → Y Z

• Decomposition: If X → Y Z, then X → Y and X → Z

• Pseudotransitivity: If X → Y and W Y → Z, then W X → Z

• Composition: If X → Y and Z → W, then XZ → Y W

**X closure**

The set of all those attributes which can be functionally determined from an attribute set is called as a closure of that attribute set.

1. A → B

2. B → C

3. BC → D

Then we can compute the closure of A with respect to M in the following way:

i A → A ( by reflexivity rule )

ii A → AB ( by (i) and 1 )

iii A → ABC ( by (ii), 2, and transitivity rule )

iv A → ABCD ( by (iii), and 3 ) Therefore, A + = ABCD.

Find X closure of A, D, B, E: R(A, B ,C, D, E)

F={A->D, D->B, B->C, E->B}

**Find the X closure**

1. R(ABCDE} F={A->D, D->B, B->C, E->B}



1. R(ABCDE) F={A->BC, CD->E, B->D, E->A}

**What is the minimal cover**

3 old sibling sisters

Wealth,status->spouse, car

Wealth->status

Status->spouse

Hard times create strong men

Strong men create good times

Good times create weak men

Weak men create hard times

Add strong women create good times

Add strong men and women create good times

A canonical cover is "allowed" to have more than one attribute on the right hand side. A minimal cover cannot.

1. Singleton RHS
2. No extraneous attributes on the left hand side (Do Closure on one and see if the other ones shows up)
3. Remove all trivial functional dependencies such as duplicates
4. No Redundancy (Do X closure and see if the RHS comes up)

In essence you are going to do Union simpliciation LHS simplification, getting rid of duplicates and getting rid of redundacies

Example 1

We have the following functional dependencies (F):

• A → BC

• B → C

we want to know if we can simplify to the following (H):

• A → B

• B → C

Example 2

1. E → GS

2. G → S

Using Decomposition

1. E → G

2. G → S

3. E → S

Further, recognizing the transitive rule

Simplifying RHS, we get:

1. E → G 2. G → S

Example 3

We have the following functional dependencies (F) for R (A,B,C)

• AB → C

• A → B

we want to know if we can simplify to the following (H):

• A → C

• A → B

Simplify the following:

1. ABC->D



AB->C

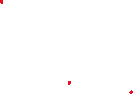


BC->A

B->A

A->C

1. a->bc,



b->c

a->b



Ab->c



1. a->bc

b->ac

c->ab



1. A → BC

B → C

A → B

AB → C



1. A→B,

ABCD→E

EF→GH

ACDF→EG

Come up with the minimal cover on this from top to bottom and bottom to top and see if F Covers G

1. A-> B

B-> C

C-> A

C-> B

B-> A

A-> C

**Does one functional dependency cover another**

Example: R(A,B,C) Are these functional dependencies the same

In looking at F, using the closure of LHS in a functional dependency, try to derive the RHS using the functional dependencies from G. If you can derive everything from F then reverse the process with G. In looking at G, using the closure of LHS in a functional dependency, try to derive the RHS using the functional dependencies from F

F=(a->b, b->c, c->a)

G=(c->b, b->a, a->c)

**Find all the keys**

**Rules:**

1. Come up with the minimal cover
2. Use the Left Middle Right model to identify which attributes appear only on the LHS (L), only on the RHS (R) and both LHS and RHS (M)
3. If in the left column, it Is definitely part of the Candidate key and is prime
4. If in the right column, it is definitely not part of the candidate key and is nonprime
5. If in the middle, then it may or may not be part of the Candidate key. If it is, then it is prime otherwise it is nonprime
6. Whatever column that is part of the relation that does not appear in any functional dependency is definitely part of the candidate key.

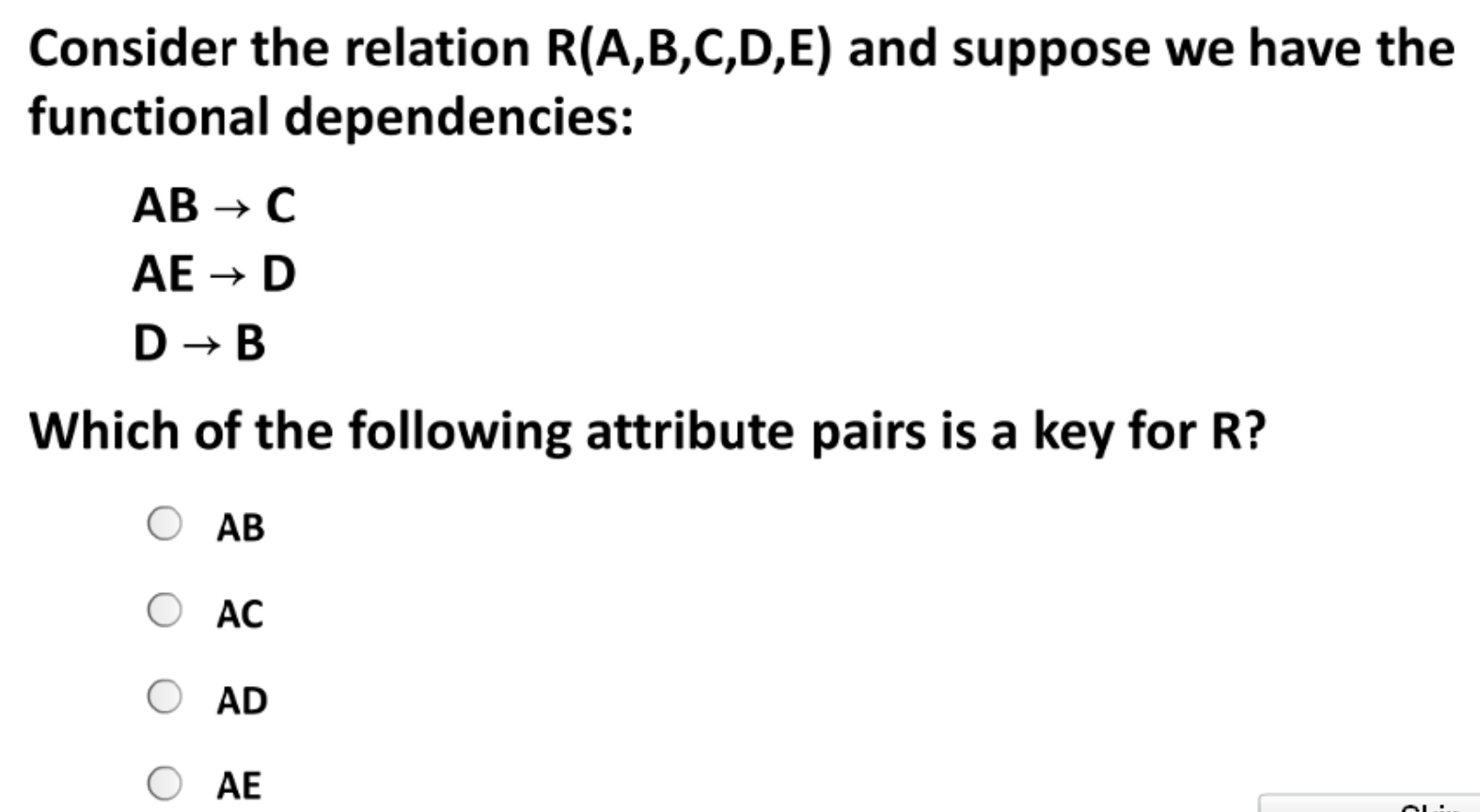
**Try these**

1. R(A,B,C,D) F={ A->B, C->D}
2. R(A,B,C,D) F={ AB->C, B->A }
3. R(A, B, C, D) F={ A->B B->C C->D CD->B BC->A}
4. R(A,B,C,D,E,F} F={ DF->C, BC->F, E->A , ABC->E} DB CEFA DB FCAE

A picture containing text

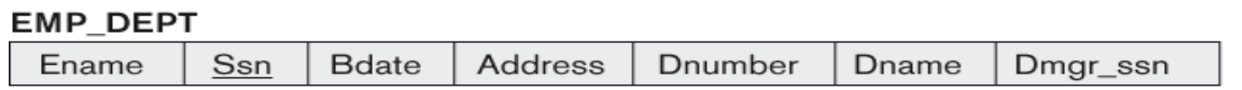
Description automatically generated







**Normal forms**



Ssn->Ename, Bdate, Address, Dnumber

Dnumber-> Dname, Dmgr\_ssn

What are the FDs for the following Relation

**R1(ssn, class\_id, class description, grade)**

Ssn, class\_id-> grade

Classid->class description

1st normal form

1. The information in each field must be atomic (single values)
2. Columns should hold dissimilar information (this means no repeating groups, no multivalues)
3. There has to be a primary key

Example:

John James, java, C should be decomposed

111, John, James, java

111, John James, C

2nd normal form

* + Find candidate keys, Identify prime and non-prime attributes
  + Partial dependency is not allowed,
    - part of candidate key in the FDs is determining non-prime attribute

Example:

abc->d abc is a candidate key, abc is prime, d is non-prime

a->d is a partial dependency because a is a partial key.

d is fully functional on abc. We say fully because we have a composite key as opposed to just functional.



3rd normal form

* + Find candidate keys, Identify prime and non-prime attributes
  + x->a either x should be a superkey or a should be a prime attribute for every functional dependency

Example:

**No Transitive dependency**

A->B and B->C then A->C (This would not be a normal form because B is not a superkey or C is not a prime

Stid name deptID deptName

11 John 1 comp

12 john 1 comp



13 jim 2 bus

Stid, name, deptID, deptName

Stdid->name

Stdid->deptID

Deptid->deptname

Stdid->departmentname

When you have a column that is dependent on another non key item and the nonkey item is dependent on the key. In other words, there cannot be any kind of interdependency among nonkey attributes.

BCNF normal form

* + Find candidate keys, Identify prime and non-prime attributes
  + x->a x is superkey

**Summary**

Come up with the minimal cover, find the candidate keys, prime and non prime

|  |  |
| --- | --- |
| 2nd normal form | Partial candidate keys cannot determine functionally determine non primes  Example: ab is candidate key, c is non prime  ab->c  a->c  or  b->c  The 2nd and 3rd Functional dependencies are partial keys that are determining nonprime |
| 3rd Normal form | For all the functional dependencies, one of two conditions must be met:   1. all the LHS must be candidate keys   or   1. all the attributes on the RHS must be prime |
| BCNF | For all the functional dependencies, all the LHS must be candidate keys |

What normal form?



1. R(A,B,C) A->B, B->C



1. R(A,B,C,D) AB->CD, C->B
2. R(A,B,C,D) AB->C, B->C, C->B
3. R(A,B,C,D) A->B, B->C, C->D, D->A
4. R(A,B,C,D) AB->C, C->D



1. R(A,B,C) AB->C, C->A
2. EmpDept(ssn, depid, dname) ssn->deptid ssn->dname
3. Empproj( ssn, p#, fname, pname) ssn, p#->fname, pname ssn->fname

Text

Description automatically generated

**Second normal form**

An entity is in *second normal form* if each attribute that is not in the primary key provides a fact that depends on the entire key. A violation of the second normal form occurs when a non-primary key attribute is a fact about a subset of a composite key. For example, an inventory entity records quantities of specific parts that are stored at particular warehouses. The following figure shows the attributes of the inventory entity.Figure 2. A primary key that violates second normal form

Begin figure description. This figure shows a primary key that violates the second normal form. End figure description.

Here, the primary key consists of the PART and the WAREHOUSE attributes together. Because the attribute WAREHOUSE\_ADDRESS depends only on the value of WAREHOUSE, the entity violates the rule for second normal form. This design causes several problems:

* Each instance for a part that this warehouse stores repeats the address of the warehouse.
* If the address of the warehouse changes, every instance referring to a part that is stored in that warehouse must be updated.
* Because of the redundancy, the data might become inconsistent. Different instances could show different addresses for the same warehouse.
* If at any time the warehouse has no stored parts, the address of the warehouse might not exist in any instances in the entity.

To satisfy second normal form, the information in the figure above would be in two entities, as the following figure shows.

Figure 3. Two entities that satisfy second normal form

Begin figure description. This figure shows two entities that satisfy the second normal form. End figure description.

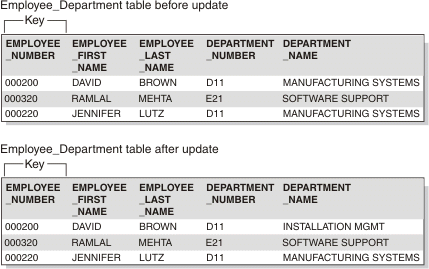
**Third normal form**

An entity is in *third normal form* if each non-primary key attribute provides a fact that is independent of other non-key attributes and depends only on the key.

A violation of the third normal form occurs when a non-primary attribute is a fact about another non-key attribute. For example, the first entity in the following figure contains the attributes EMPLOYEE\_NUMBER and DEPARTMENT\_NUMBER. Suppose that a program or user adds an attribute, DEPARTMENT\_NAME, to the entity. The new attribute depends on DEPARTMENT\_NUMBER, whereas the primary key is on the EMPLOYEE\_NUMBER attribute. The entity now violates third normal form.

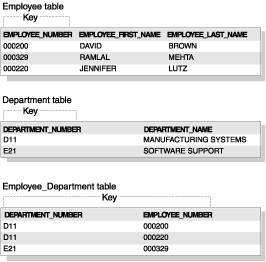
Changing the DEPARTMENT\_NAME value based on the update of a single employee, David Brown, does not change the DEPARTMENT\_NAME value for other employees in that department. The updated version of the entity in the following figure illustrates the resulting inconsistency. Additionally, updating the DEPARTMENT\_NAME in this table does not update it in any other table that might contain a DEPARTMENT\_NAME column.

Figure 4. The update of an unnormalized entity. Information in the entity has become inconsistent.



You can normalize the entity by modifying the EMPLOYEE\_DEPARTMENT entity and creating two new entities: EMPLOYEE and DEPARTMENT. The following figure shows the new entities. The DEPARTMENT entity contains attributes for DEPARTMENT\_NUMBER and DEPARTMENT\_NAME. Now, an update such as changing a department name is much easier. You need to make the update only to the DEPARTMENT entity.

Figure 5. Normalized entities: EMPLOYEE, DEPARTMENT, and EMPLOYEE\_DEPARTMENT



BCNF

Table

Description automatically generated

St-id, subject -> professor

Professor -> subject //violation because professor is not a key

A professor can teach only one subject

Graphical user interface, application

Description automatically generated

**Fourth normal form**

An entity is in fourth normal form if no instance contains two or more independent, multivalued facts about an entity.

For example, Consider the EMPLOYEE entity. Each instance of EMPLOYEE could have both SKILL\_CODE and LANGUAGE\_CODE. An employee can have several skills and know several languages. Two relationships exist, one between employees and skills, and one between employees and languages. An entity is not in fourth normal form if it represents both relationships, as the following figure shows.

Figure 6. An entity that violates fourth normal form

Begin figure description. This figure shows an entity that violates the fourth normal form. End figure description.

Instead, you can avoid this violation by creating two entities that represent both relationships, as the following figure shows.

Figure 7. Entities that are in fourth normal form

Begin figure description. This figure shows entities that are in the fourth normal form. End figure description.

If, however, the facts are interdependent (that is, the employee applies certain languages only to certain skills) you should not split the entity.

You can put any data into fourth normal form. A good rule to follow when doing logical database design is to arrange all the data in entities that are in fourth normal form. Then decide whether the result gives you an acceptable level of performance. If the performance is not acceptable, denormalizing your design is a good approach to improving performance.