Normalization

20 October 2024 11:39

Normalization in database management system is the process of organizing data in a database to minimize redundancy and improve data integrity. It involves dividing a database into two or more tables and defining relationship between them to reduce duplication and ensure data dependencies make sense.

Problems caused by redundancy.

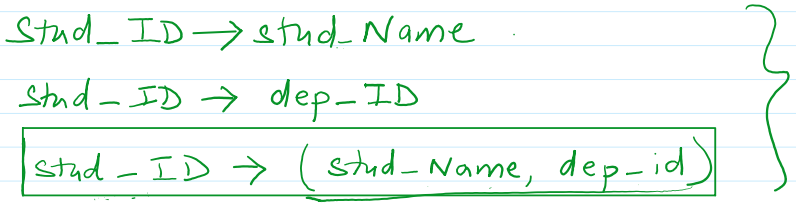
Storing the same information redundantly, that is, in more than one places within a database can lead to several problems.

- **Redundant storage:** Some information is stored repeatedly.
- **Update anomalies:** If one copy of such repeated data is updated and. inconsistency is created unless all copies are similarly updated.
- **Insertion anomalies:** It may not be possible to store some information unless some other information is stored as well.
- **Deletion anomalies:** It may not be possible to delete some information without losing some other information as well.

Functional Dependencies (FD)

A functional dependency in a database refers. to a relationship between two sets of attributes or columns within a table. Specifically of functional dependency, occurs when one attribute or a group of attributes uniquely determines another attribute. If knowing the values of one attribute allows you to know the value of another, then the 2nd attribute is set to be functionally dependent on the first.

Notation: A->B {Attribute A determines attribute B}



1. **Determinant:** The attribute or set of attributes on the left. On the left side of the functional dependency is called the determinant. It uniquely determines the value of the other attribute.
2. **Dependent:** The attribute on the right side. is called the dependent attribute. Its value is determined by the determinant.

TABLE 6.1

A SAMPLE REPORT LAYOUT

PROJECT NUMBER	PROJECT NAME	EMPLOYEE NUMBER	EMPLOYEE NAME	JOB CLASS	CHARGE/ HOUR	HOURS BILLED	TOTAL CHARGE
15	Evergreen	103	June E. Arbough	Elec. Engineer	$ 84.50	23.8	$ 2,011.10
		101	John G. News	Database Designer	$105.00	19.4	$ 2,037.00
		105	Alice K. Johnson *	Database Designer	$105.00	35.7	$ 3,748.50
		106	William Smithfield	Programmer	$ 35.75	12.6	$ 450.45
		102	David H. Senior	Systems Analyst	$ 96.75	23.8	$ 2,302.65
				Subtotal			**$10,549.70**
18	Amber Wave	114	Annelise Jones	Applications Designer	$ 48.10	24.6	$ 1,183.26
		118	James J. Frommer	General Support	$ 18.36	45.3	$ 831.71
		104	Anne K. Ramoras *	Systems Analyst	$ 96.75	32.4	$ 3,134.70
		112	Darlene M. Smithson	DSS Analyst	$ 45.95	44.0	$ 2,021.80
				Subtotal			**$ 7,171.47**
22	Rolling Tide	105	Alice K. Johnson	Database Designer	$105.00	64.7	$ 6,793.50
		104	Anne K. Ramoras	Systems Analyst	$96.75	48.4	$ 4,682.70
		113	Delbert K. Joenbrood *	Applications Designer	$48.10	23.6	$ 1,135.16
		111	Geoff B. Wabash	Clerical Support	$26.87	22.0	$ 591.14
		106	William Smithfield	Programmer	$35.75	12.8	$ 457.60
				Subtotal			**$13,660.10**
25	Starflight	107	Maria D. Alonzo	Programmer	$ 35.75	24.6	$ 879.45
		115	Travis B. Bawangi	Systems Analyst	$ 96.75	45.8	$ 4,431.15
		101	John G. News *	Database Designer	$105.00	56.3	$ 5,911.50
		114	Annelise Jones	Applications Designer	$ 48.10	33.1	$ 1,592.11
		108	Ralph B. Washington	Systems Analyst	$ 96.75	23.6	$ 2,283.30
		118	James J. Frommer	General Support	$ 18.36	30.5	$ 559.98
		112	Darlene M. Smithson	DSS Analyst	$ 45.95	41.4	$ 1,902.33
				Subtotal			**$17,559.82**
				Total			**$48,941.09**

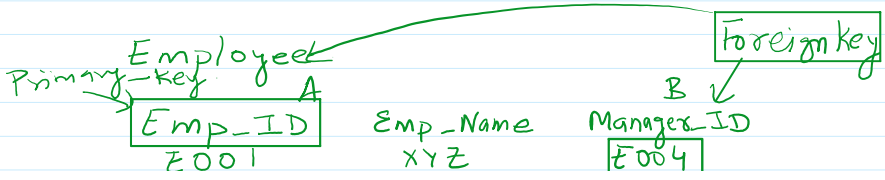
Note: A * indicates the project leader.

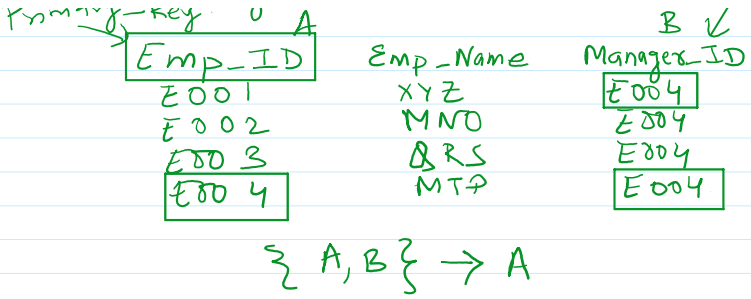
FUNCTIONAL DEPENDENCE CONCEPTS

CONCEPT	DEFINITION
Functional dependence	The attribute *B* is fully functionally dependent on the attribute *A* if each value of *A* determines one and only one value of *B*. Example: PROJ_NUM → PROJ_NAME (read as *PROJ_NUM functionally determines PROJ_NAME*) In this case, the attribute PROJ_NUM is known as the determinant attribute, and the attribute PROJ_NAME is known as the dependent attribute.
Functional dependence (generalized definition)	Attribute *A* determines attribute *B* (that is, *B* is functionally dependent on *A*) if all (generalized definition) of the rows in the table that agree in value for attribute *A* also agree in value for attribute *B*.
Fully functional dependence (composite key)	If attribute *B* is functionally dependent on a composite key *A* but not on any subset of that composite key, the attribute *B* is fully functionally dependent on *A*.

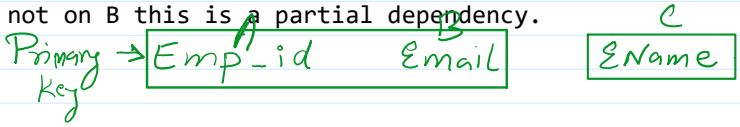
Types of functional dependencies:

1. **Trivial Functional dependency:** Of function dependency is trivial if the dependent attribute is a subset of the determinant. For example, {A,B}->A is a trivial functional dependency because knowing both A and B trivially determines A.
2. **Non-trivial functional dependency**: If the dependent attribute is not a subset of a determinant. it is nontrivial, functional dependency. For example, A->B is non-trivial if is not part of A.





3. **Partial functional dependency:** In the case of composite keys. a partial dependency exist when. an attribute depends on. only part of the composite key rather than the whole key. For example, in a table with a composite key (A,B), if C depends only on A but not on B this is a partial dependency.



4. **Transitive functional dependency:** A transitive dependency occurs when an attribute depends indirectly on a determinant. For instance, if A->B and B->, then A->C is a transitive functional dependency.

Types of Normal Forms:

NORMAL FORMS	
NORMAL FORM	**CHARACTERISTIC**
First normal form (1NF)	Table format, no repeating groups, and PK identified
Second normal form (2NF)	1NF and no partial dependencies
Third normal form (3NF)	2NF and no transitive dependencies
Boyce-Codd normal form (BCNF)	Every determinant is a candidate key (special case of 3NF)
Fourth normal form (4NF)	3NF and no independent multivalued dependencies

What is Normalization?
It is a process for evaluating and correcting table structures to minimize data redundancies there. reducing the likelihood of data anomalies.

Normalization walks through a series of stages called normal forms. The. first three stages are described as first normal form or (1NF), second normal form (2NF) and third normal form 3NF.
From a structural point of view, 2NF is better than 1NF and 3NF is better than 2NF.

Denormalization:
Produce a lower normal form which is a 3NF will be converted to a 2NF through denormalization. A successful design must also consider end user demand for fast performance. Therefore, you will occasionally be expected to de-normalize some portion of database design in order to meet performance requirement.

The Normalization Process:
We will learn how to use normalization to produce a set of normalized table to store the data that will be used to generate the required information. The objective of normalization is to ensure that each table conforms to the concept of well-formed relation, that is, tables that have the following characteristics.

- Each payable represents a single subject for example. a course table will contain only data that directly pertains. to courses. similarly, a student table will contain

only student data

- No data item will be unnecessarily stored in more than one table (In short, tables have minimal. controlled redundancy.) The reason for this requirement is to ensure that the data are updated in one place.
- All non-prime attributes in a table are dependent on the primary key. The reason for this requirement is to ensure that the data are uniquely identifiable by a primary key value.
- Each table is void of insertion update or deletion anomalies. This is to ensure the integrity and consistency of the data.

Conversion to First Normal Form(1NF)

Step-1:: Eliminate the Repeating Groups

Start by presenting the data in tabular format where each cell has a single value and there are no repeating groups. A repeating group derives its name from the fact that a group of multiple entries of the same type can exist for any single key attribute occurrence. To eliminate the repeating groups, eliminate the nulls by making sure that each repeating group attribute contains an appropriate data value.

FIGURE 6.2 A TABLE IN FIRST NORMAL FORM

Table name: DATA_ORG_1NF **Database name: Ch06_ConstructCo**

PROJ_NUM	PROJ_NAME	EMP_NUM	EMP_NAME	JOB_CLASS	CHG_HOUR	HOURS
15	Evergreen	103	June E. Arbough	Elect. Engineer	84.50	23.8
15	Evergreen	101	John G. News	Database Designer	105.00	19.4
15	Evergreen	105	Alice K. Johnson *	Database Designer	105.00	35.7
15	Evergreen	106	William Smithfield	Programmer	35.75	12.6
15	Evergreen	102	David H. Senior	Systems Analyst	96.75	23.8
18	Amber Wave	114	Annelise Jones	Applications Designer	48.10	24.6
18	Amber Wave	118	James J. Frommer	General Support	18.36	45.3
18	Amber Wave	104	Anne K. Ramoras *	Systems Analyst	96.75	32.4
18	Amber Wave	112	Darlene M. Smithson	DSS Analyst	45.95	44.0
22	Rolling Tide	105	Alice K. Johnson	Database Designer	105.00	64.7
22	Rolling Tide	104	Anne K. Ramoras	Systems Analyst	96.75	48.4
22	Rolling Tide	113	Delbert K. Joenbrood *	Applications Designer	48.10	23.6
22	Rolling Tide	111	Geoff B. Wabash	Clerical Support	26.87	22.0
22	Rolling Tide	106	William Smithfield	Programmer	35.75	12.8
25	Starflight	107	Maria D. Alonzo	Programmer	35.75	24.6
25	Starflight	115	Travis B. Bawangi	Systems Analyst	96.75	45.8
25	Starflight	101	John G. News *	Database Designer	105.00	56.3
25	Starflight	114	Annelise Jones	Applications Designer	48.10	33.1
25	Starflight	108	Ralph B. Washington	Systems Analyst	96.75	23.6
25	Starflight	118	James J. Frommer	General Support	18.36	30.5
25	Starflight	112	Darlene M. Smithson	DSS Analyst	45.95	41.4

Step-2::Identify the primary key

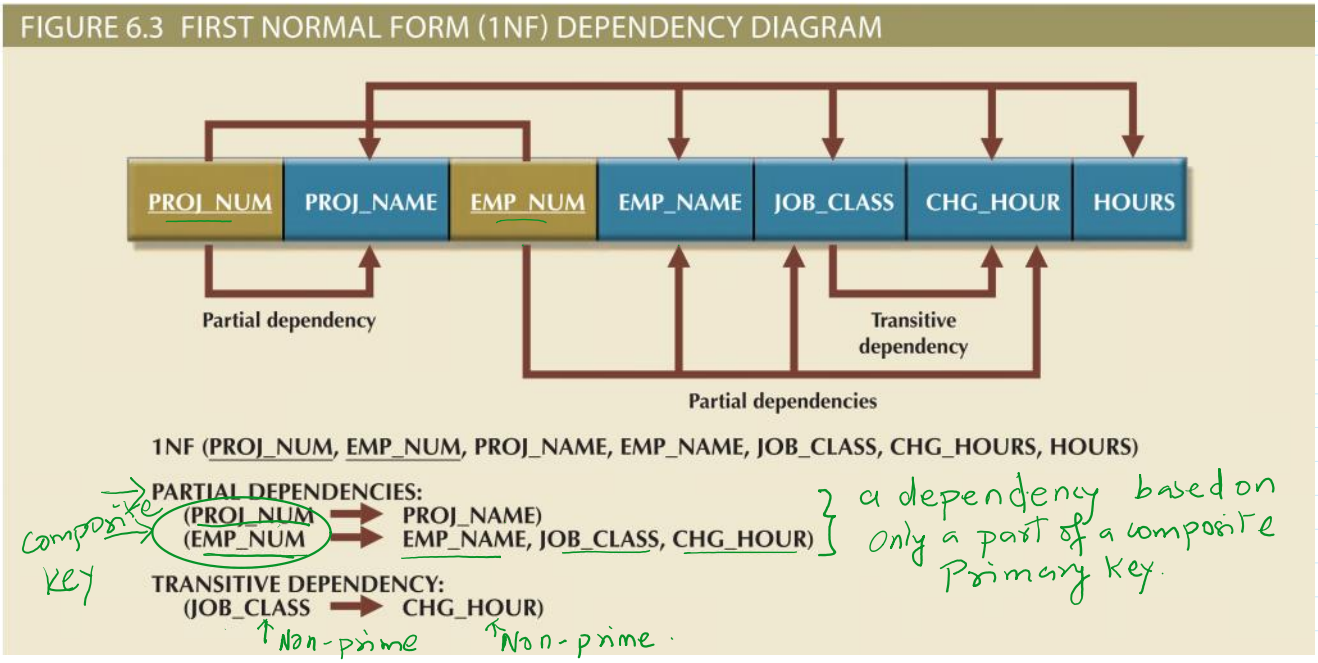
Even casual observers will note that PROJ_NUM is not an adequate primary key because the project number does not uniquely identify all of the remaining entity (row) attributes. To maintain a proper primary key that will uniquely identify any attribute value. The new key must be. composed of combination of a PROJ_NUM and EMP_NUM.

Step-3::Identify All Dependencies:

The identification of the primary key in step 2 means that you have already identified the following dependencies:

* PROJ_NUM, EMP_NUM → PROJ_NAME, EMP_NAME, JOB_CLASS. CHG_HOUR, HOURS.
* PROJ_NUM → PORJ_NAME
* EMP_NUM → (EMP_NAME, JOB_CLASS, CHG_HOUR)
* JOB_CLASS → CHG_HOUR.

FIGURE 6.3 FIRST NORMAL FORM (1NF) DEPENDENCY DIAGRAM



Step-1: Eliminate the repeating Groups.

Step-2: Identify the Primary key

Step-3: Identify all dependencies.

The first normal form describes the tabular format in which:

- All of the key attributes are defined.
- There are no repeating groups in the table. In other words, each row or column intersection contains 1 and only 1 value, not a set of values.
- All attributes are dependent on the primary key.

The problem with the first normal form table structure is that it contains partial dependencies, while, partial dependencies are sometimes used for performance reason, they should be used with caution.

First normal or 1NF

A relation in 1NF if:

- All values in each column are atomic, meaning they are indivisible. (no repeating groups or arrays.)
- Each column contains values of single type.
- Each row uniquely identifies an instance of a data.

↑ object

In 1NF tables might. still contain redundancy and partial dependencies, which is addressed by advancing to higher normal forms.

Second Normal Form (2NF)

A relation is 2NF if:
1. It is already in 1NF.
2. It has no partial dependencies, meaning:
a. Every non-key attribute must depend on the primary key not just part of it.

A **partial dependency** occurs when a non-key attribute depends on only part of a composite primary key, rather than the entire primary key. This issue is only present in tables where the primary key is composite (that is made up of more than one attribute) if the primary key is simple, or a single attribute, then the table in 1NF is automatically in 2NF.

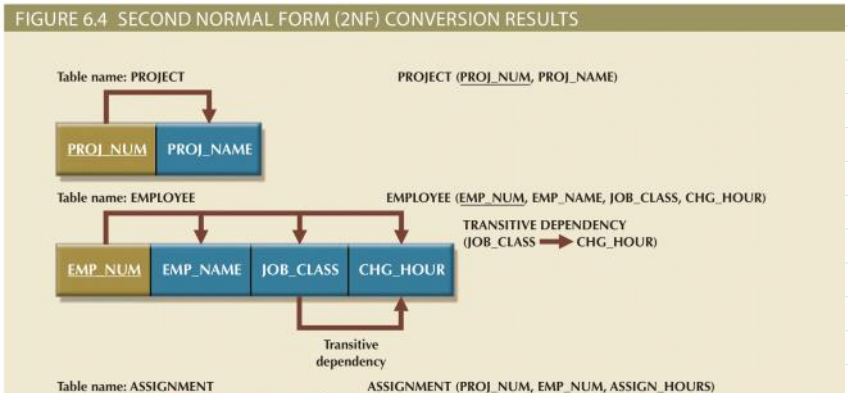
Steps to convert 1NF to 2NF

1. **Identify the primary key and non-key attributes:**
   i. **First, determine the primary key for the table, especially. if it is a composite key**
   ii. **Identify which attributes or columns are non-key attributes. Non key attributes are columns that are the part of the primary key.**
2. **Check for partial dependencies:**
   i. **For each non key attribute verify if it depends on the entire primary key.**
   ii. **If a non-key attribute only depends on a part of a composite key, then a partial dependency exists, and the table is not in 2NF.**
3. **Eliminate partial dependencies:**
   i. **If there are partial dependencies, decompose the table into two or more tables to eliminate them.**
   ii. **Each new table should have its own primary key and each nonkey attribute should fully depend on that key.**
4. **Re-establish relationship using foreign keys:**
   i. **If you have decomposed the original table into multiple tables add foreign keys as necessary to maintain relationship between them.**

To construct the revised dependency diagram, write each key component on a separate line and then write the original (composite) key on the last line. For example:
PROJ_NUM
EMP_NUM
PROJ_NUM EMP_NUM
Each component will become the key in a new table. In other words, the original table is now divided into three tables (PROJECT, EMPLOYEE, and ASSIGNMENT).

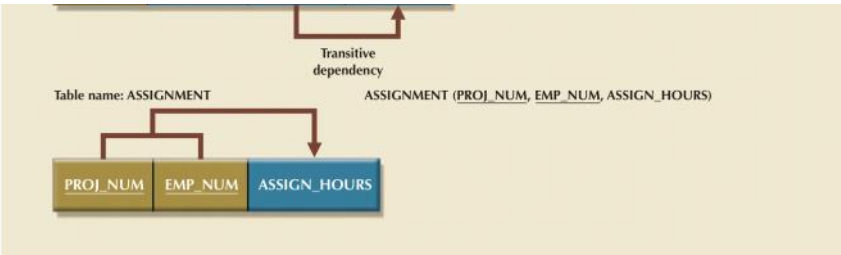
PROJECT (PROJ_NUM, PROJ_NAME)
EMPLOYEE (EMP_NUM, EMP_NAME, JOB_CLASS, CHG_HOUR)
ASSIGNMENT (PROJ_NUM, EMP_NUM, ASSIGN_HOURS)

Because the number of hours spent on each project by each employee is dependent on both PROJ_NUM and EMP_NUM in the ASSIGNMENT table, you leave those hours in the ASSIGNMENT table as ASSIGN_HOURS. Notice that the ASSIGNMENT table contains a composite primary key composed of the attributes PROJ_NUM and EMP_NUM. Notice that by leaving the determinants in the original table as well as making them the primary keys of the new tables, primary key/foreign key relationships have been created. For example, in the EMPLOYEE table, EMP_NUM is the primary key. In the ASSIGNMENT table, EMP_NUM is part of the composite primary key (PROJ_NUM, EMP_NUM) and is a foreign key relating the EMPLOYEE table to the ASSIGNMENT table.



No - partial dependencies are present

Hence, this is 2NF



Second Normal Form (2NF)

A relation is in 2NF if:

1. It is already in first normal form, or 1NF.

2. It has no partial dependencies, which means.:

- **Every non key attribute be fully functionally dependent on the entire primary key.**

In simpler terms, there should be no non key attribute depending on a part of a composite key. However, a table in 2NF can still have transitive dependency, which can, lead to redundancy, transitive dependencies are what 3NF aims to eliminate.

Third Normal Form (3NF)

A relation is in 3NF if:

1. It is already in 2NF.

2. It has no transitive dependencies, meaning:

- **No non-key attribute should depend on another non-key attribute.**

A **transitive dependency** occurs when a non-key attribute depends indirectly on the primary key through another non key attribute. If this happens, it usually means that certain non-key attributes depend on one another, rather than directly on the primary key, **creating potential redundancy** and **update anomalies.**

Steps to Convert 2NF to 3NF

1. Identify the primary key and non-key attributes:

- **i. Determine the primary key for the table.**
- **ii. List out the non-key attributes to check for any dependencies between them.**

2. Identify transitive dependencies:

- **i. Look for attributes that depend on other non-key attribute, rather than directly on the primary key.**
- **ii. If you find a non-key attribute that is functionally dependent on another non-key attribute, then transitive dependency exist.**

3. Eliminate transitive dependencies:

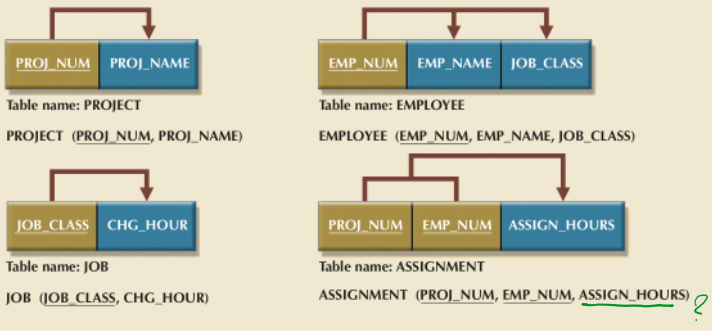
- **i. Decompose the table by creating separate tables for each transitive dependency.**
- **ii. Ensure that in each new table, non-key attributes depend only on the primary key of the table.**

4. Reestablish relationship using foreign keys:

- **i. If you decompose the original table into multiple tables, use foreign keys to maintain the relationship between them as needed.**

Draw a new dependency diagram to show all of the tables you have defined in Steps 1 and 2. Name the table to reflect its contents and function. In this case, JOB seems appropriate. Check all of the tables to make sure that each table has a determinant and that no table contains inappropriate dependencies. When you have completed these steps, you will see the results in Figure 6.5

FIGURE 6.5 THIRD NORMAL FORM (3NF) CONVERSION RESULTS



In other words, after the 3NF conversion has been completed, your database will contain four tables:

PROJECT (PROJ_NUM, PROJ_NAME)
EMPLOYEE (EMP_NUM, EMP_NAME, JOB_CLASS)
JOB (JOB_CLASS, CHG_HOUR)
ASSIGNMENT (PROJ_NUM, EMP_NUM, ASSIGN_HOURS)

Note that this conversion has eliminated the original EMPLOYEE table's transitive dependency. The tables are now said to be in third normal form (3NF)

Improving the design

1. Evaluate PK Assignments: Each time a new employee is entered into the employee table. A JOB_CLASS value. must be entered. Unfortunately, it is too easy to make the data entry errors that lead to referential integrity violations. For example, entering DBdesigner instead of "database designer" for the JOB_CLASS attribute in the employee table will trigger such a violation. Therefore it would be better to add a job code attribute to create a unique identifier. The addition of a JOB_CODE attribute produce the following dependency:
   JOB_CODE ->JOB_CLASS, CHG_HOURS
   If you assume that the job _code is a proper primary key, this new attribute does produce the following dependency:

   JOB_CLASS->CHG_HOURS

   However, this dependency is not a transitive dependency because the determinant is a candidate key. Further, the presence of job _code greatly decreases the likelihood keys, JOB_CODE and JOB_CLASS. In this case, JOB_CODE is chosen primary key as well as a surrogate key. A surrogate key, as you should recall, is an artificial primary key introduced by the designer with the purpose of simplifying the assignment of primary keys to tables. Surrogate keys are usually numeric. They are often generated automatically by the dbms they are free of semantic content(They have no special meaning) and they are hidden from the end users.

2. Evaluate Naming Conventions: CHG_HOUR will be changes to JOB_CHG_HOUR to indicate its association with the JOB table. In addition, the attribute name JOB_CLASS does not quite describe entries such as System Analyst, Database Designer and so on; the label JOB_DESCRIPTION fits the entries better, Also, you might have noticed that HOURS was changed to a ASSIGN_HOURS in the conversion from 1NF to 2NF. That change lets you to associate the hours worked with the ASSIGNMENT table.

3. Refine Attribute Atomicity: It is generally good practice to pay attention to the atomicity requirement. An atomic attribute is one that cannot be further subdivided. Such an attribute is said to display atomicity. Clearly, the use of EMP_NAME in the EMPLOYEE table is not atomic, because EMP_NAME can be decomposed into a last name, a first name and an initial, by improving the degree of atomicity, you can also gain querying flexibility. For example, if

you use EMP_LNAME, EMP_FNAME and EMP_INITIAL, you can easily generate phone list by sorting last name, first names and initials. Such a task would be very difficult if name components were within a single attribute. In general, designers prefer to use simple, single valued attribute as indicate by the business rules and processing requirements.

4. **Identify New Attributes: If the EMPLOYEE table were used in real world environment, several other attributes would have to be added. For example, year-to-date, gross salary payments, Social Security payments and Medicare payments would be desirable. And employee hired date attribute EMP_HIREDATE could be used to track an employee's job longevity, and it could serve as a basis for awarding bonuses to long term employees. And for other moral enhancing measures the same principle must be applied to all other tables in your design.**

5. **Refine Primary Key as Required for Data Granularity: Granularity refers to the level of detail represented by the values stored in a table's row. Data stored at its lowest level of granulating is set to be atomic data. In figure 6.5 the ASSIGNMENT table in three NF uses the ASSIGN_HOUR attribute to represent the hours worked by a given employee on a given project. However, are those values recorded at their lowest level of granularity? In other words does ASSIGN_HOUR represents the hourly total, daily total, weekly total, monthly total or yearly total clearly ASSIGN_HOURS requires more careful definition in this case, the relevant question would be as follows. For what time frame? our day, week, month and so on - Do you want to record the ASSIGN_HOUR data?**
   For example, assume that the combination of EMP_NUM and PROJ_NUM is acceptable(composite) primary key in the ASSIGNMENT table. That primary key is useful in representing only the total number of hours and employee worked on projects since its start. using a surrogate primary key, such as a sign provides lower granularity and yields greater flexibility. For example, assume that EMP_NUM and PROJ_NUM Combination is used as a primary key and then an employee makes two hours worked entries in the ASSIGNMENT table. That action violates the entity integrity requirement. Even if you add the ASSIGN_DATE as a part of a composite primary key an entity integrity violation is still generated if employee makes two or more entries for the same project on the same day. The employee might have worked on a project for few hours in the morning, and then worked on it again later in the day. The same day entry yields no problem when a ASSIGN_NUM is used as a primary key.

   NOTE: *An ideal database design, the level of. desired granularity would be determined during the conceptual design or while the requirements were being gathered. However, as you have already seen, many database design involves the refinement of existing data requirements thus triggering design modification in a real world environment, changing granularity requirements might dictate changes in primary key selection and those changes might ultimately require the use of surrogate keys.*

6. **Maintaining Historical Accuracy:** Writing the job charge per hour into the **ASSIGNMENT** table is crucial to maintaining the history accuracy of the table's data. It would be appropriate to name this attribute ASSIGN_CHG_HOUR although this attribute would appear to have the same value as JOB_CHG_HOUR. This is truly only if the JOB_CHG_HOUR value remains the same forever. It is reasonable to assume that job charge per hour will change over time. However, suppose that the charges to each project were calculated and billed by multiplying the hours worked from the **ASSIGNMENT** table by the charge per hour from the **JOB** table. Those charges would always show the current charge per hour stored in the **JOB** table, rather than charge per hour. That was in effect at the time of the assignment.

7. **Evaluate Using Derived Attributes: Finally, you can use a derived attribute in ASSIGNMENT table to store the actual charge made to the project. That derived attribute named ASSIGN_CHARGE is a result of multiplying a ASSGIN_HOUR multiply by ASSIGN_CHG_HOUR. This creates a transitive dependencies such that: (ASSIGN_CHARGE + ASSIGN_HOURS) -> ASSIGN_CHG_HOUR**

From a system functionality point of view, such derived attribute values can be calculated when they are needed to write reports or invoices. However, storing the derived attribute in the table makes it easy to write the application software to produce the desired results. Also, if many transaction must be reported and Oregon summarized the availability of the derived attribute will save reporting time.

FIGURE 6.6 THE COMPLETED DATABASE

Table name: PROJECT

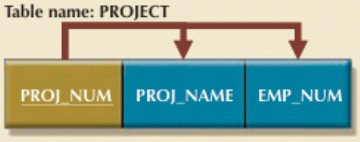


Table name: JOB Database name: Ch06_ConstructCo

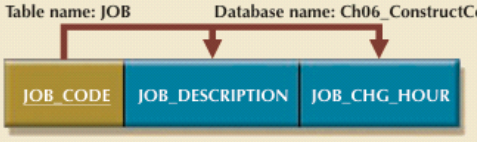


Table name: PROJECT

PROJ_NUM	PROJ_NAME	EMP_NUM
15	Evergreen	105
18	Amber Wave	104
22	Rolling Tide	113
25	Starflight	101

Table name: JOB

JOB_CODE	JOB_DESCRIPTION	JOB_CHG_HOUR
500	Programmer	35.75
501	Systems Analyst	96.75
502	Database Designer	105.00
503	Electrical Engineer	84.50
504	Mechanical Engineer	67.90
505	Civil Engineer	55.78
506	Clerical Support	26.87
507	DSS Analyst	45.95
508	Applications Designer	48.10
509	Bio Technician	34.55
510	General Support	18.36

Table name: ASSIGNMENT

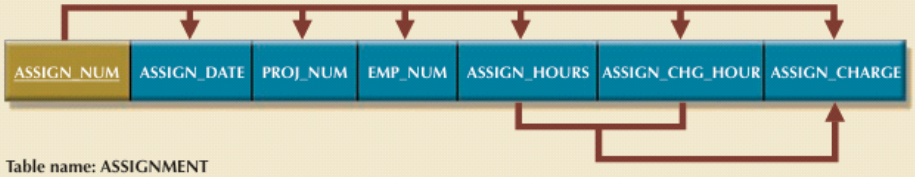


Table name: ASSIGNMENT

ASSIGN_NUM	ASSIGN_DATE	PROJ_NUM	EMP_NUM	ASSIGN_HOURS	ASSIGN_CHG_HOUR	ASSIGN_CHARGE
1001	04-Mar-16	15	103	2.6	84.50	219.70
1002	04-Mar-16	18	118	1.4	18.36	25.70
1003	05-Mar-16	15	101	3.6	105.00	378.00
1004	05-Mar-16	22	113	2.5	48.10	120.25
1005	05-Mar-16	15	103	1.9	84.50	160.55
1006	05-Mar-16	25	115	4.2	96.75	406.35
1007	05-Mar-16	22	105	5.2	105.00	546.00
1008	05-Mar-16	25	101	1.7	105.00	178.50
1009	05-Mar-16	15	105	2.0	105.00	210.00
1010	06-Mar-16	15	102	3.8	96.75	367.65
1011	06-Mar-16	22	104	2.6	96.75	251.55
1012	06-Mar-16	15	101	2.3	105.00	241.50
1013	06-Mar-16	25	114	1.8	48.10	86.58
1014	06-Mar-16	22	111	4.0	26.87	107.48
1015	06-Mar-16	25	114	3.4	48.10	163.54
1016	06-Mar-16	18	112	1.2	45.95	55.14
1017	06-Mar-16	18	118	2.0	18.36	36.72
1018	06-Mar-16	18	104	2.6	96.75	251.55
1019	06-Mar-16	15	103	3.0	84.50	253.50
1020	07-Mar-16	22	105	2.7	105.00	283.50
1021	08-Mar-16	25	108	4.2	96.75	406.35
1022	07-Mar-16	25	114	5.8	48.10	278.98
1023	07-Mar-16	22	106	2.4	35.75	85.80

FIGURE 6.6 THE COMPLETED DATABASE (CONTINUED)

Table name: EMPLOYEE Database name: Ch06_ConstructCo

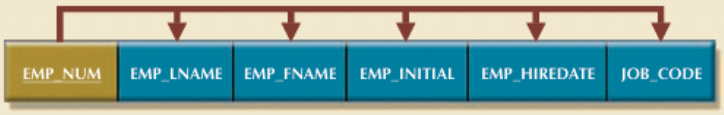


Table name: EMPLOYEE

EMP_NUM	EMP_LNAME	EMP_FNAME	EMP_INITIAL	EMP_HIREDATE	JOB_CODE
101	News	John	G	08-Nov-00	502
102	Senior	David	H	12-Jul-89	501
103	Arbough	June	E	01-Dec-97	503
104	Ramoras	Anne	K	15-Nov-88	501
105	Johnson	Alice	K	01-Feb-94	502
106	Smithfield	William		22-Jun-05	500
107	Alonzo	Maria	D	10-Oct-94	500
108	Washington	Ralph	B	22-Aug-89	501
109	Smith	Larry	W	18-Jul-99	501
110	Olenko	Gerald	A	11-Dec-96	505
111	Wabash	Geoff	B	04-Apr-89	506
112	Smithson	Darlene	M	23-Oct-95	507
113	Joenbrood	Delbert	K	15-Nov-94	508
114	Jones	Annelise		20-Aug-91	508
115	Bawangi	Travis	B	25-Jan-90	501
116	Pratt	Gerald	L	05-Mar-95	510
117	Williamson	Angie	H	19-Jun-94	509
118	Frommer	James	J	04-Jan-06	510

Boyce-Codd Normal Form(BCNF) Is a higher level of database normalization, aiming to reduce redundancy and dependency anomalies in a relational database. named after Edgar F.Codd (The creator of the relational model) and Raymond F.Boyce is a stricter version of the 3rd normal form or 3NF. BCNF ensures that every determinant in a relation is a candidate key.

1NF: Ensures that all attribute contains only atomic(indivisible) values.
2NF: Ensure that all non-key attributes are fully functionally dependent on the primary key.
3NF: Ensures that all non-key attribute and non-transitively dependent on the primary key.
BCNF: A stricter form of 3NF where every determinant must be a candidate key.

Definition of BCNF:
A relation is in BCNF if, for every functional dependency (A->B) in relation, A is a super-key.

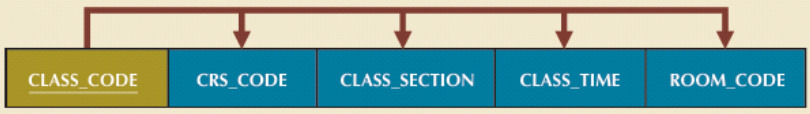
In other words"

- **A relation is in BCNF, if, whenever there is a functional dependency, X->Y, X must be a candidate key, (A minimal super key) for the table.**
- **A candidate key is an? attribute or a minimal combination of attributes that can uniquely identify a tuple(row) in a table.**
- **A super key is a set of one or more attributes that, taken together, can uniquely identify a tuple in a table.**

If a relation does not meet this condition, it can. lead to anomalies like update insertion and deletion anomalies which be BCNF helps to prevent.

Clearly, when a table contains only one candidate key, the 3NF and the BCNF are equivalent. In other words, BCNF can be violated only when the table contains more than one candidate key. in the previous normal form examples, table with 1 candidate keys were used to simplify the explanations. Remember, however, that multiple candidate keys are always possible and normalization rules focus on candidate keys, not just the primary key.

FIGURE 6.7 TABLES WITH MULTIPLE CANDIDATE KEYS



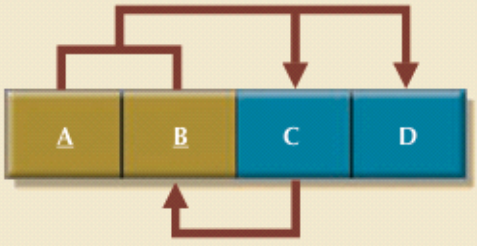
The CLASS table has two candidate keys:

- CLASS_CODE
- CRS_CODE + CLASS_SECTION

The table is in 1NF because the key attributes are defined, and all nonkey attributes are determined by the key. This is true for both candidate keys. Both candidate keys have been defined, and all of the other attributes can be determined by either candidate. The table is not 2NF because it is in 1NF. And there are no partial dependencies on either candidate keys. Since CLASS_CODE is a single attribute candidate key, the. issue of the partial dependency does not apply. However, the composite candidate key of CRS_CODE + CLASS_SECTION could potentially have a partial dependency. So 2nf must be evaluated for that candidate key. In this case, there are no partial dependencies involving the composite key. Finally, the table is in 3NF because there are no transitive dependencies. Remember, because CRS_CODE + CLASS_SECTION is a candidate key. The fact that this composite can determine the CLASS_TIME and ROOM_CODE is not a transitive dependency. Hot transitive dependency exists when a non-key attribute can determine another non key attribute and CRS_CODE + CLASS_SECTION is a key.

Most designers consider that BCNF to be a special case of 3NF. In fact, if the techniques are. Used. most table conform to the BCNF requirements once the 3NF is reached. So how can a table be in 3NF and not in BCNF? to answer that question, you must keep in mind that transitive dependency exists when a non-prime attribute is dependent on another non-prime attribute. In other words, a table is in 3NF when it is in 2NF. And there are no transitive dependencies. But what about a case in which one key attribute in the determinant of another key attribute? That condition does not violate 3NF yet it fails to meet the BCNF requirements. Because BCNF requires that every determinant in a table be a candidate key.

FIGURE 6.8 A TABLE THAT IS IN 3NF BUT NOT IN BCNF

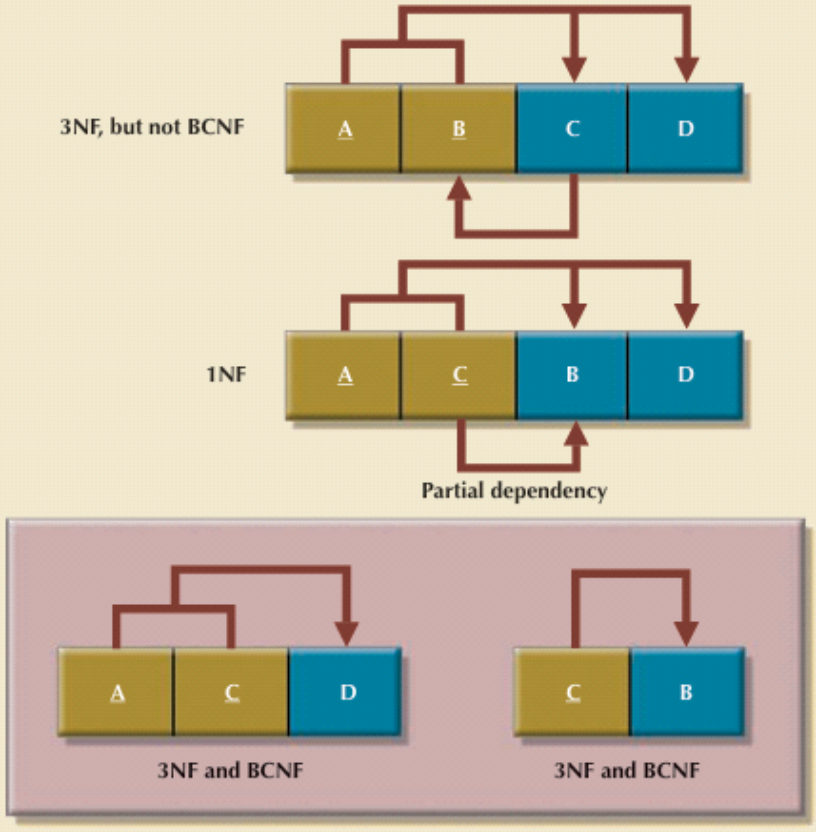


(A + B) → C, D
(A + C) → B, D
C → B

Notice that this structure has two candidate keys. (A + B) and (A + C). The table structure shown in figure 6.8 has no partial dependencies, nor does it contain transitive dependencies. (the condition C -> B Indicates that one key attribute determines part of the primary key, and that dependency is not transitive or partial because the dependent is a primary attribute.) Thus, the table structure and figures 6.8 meets the three NF requirements, although the condition C -> B causes the table to fail to meet the BCNF requirements.
To convert the table structure in figure 6.8 into a table structure that are in 3NF and in BCNF first, change the primary key to a + c. This change is

appropriate because the dependency C -> B means that C is effectively a super set of B. At this point, the table is in 1NF because it contains a partial dependency, C->B, Next, follow the standard decomposition procedure to produce the result shown in figure 6.9.

FIGURE 6.9 DECOMPOSITION TO BCNF



To see how this procedure can be applied to an actual problem Examine the sample data in tables 6.5.
Table 6.5 reflects the following conditions:

- Each CLASS_CODE Identifies a class uniquely This condition illustrates the case in which a course might generate many classes. For example, a course labeled INFS, 420 might be taught in two classes. (section.) Each identified by a unique code to facilitate registration. Does the CLASS_CODE 32456 might identify INFS 420, class section 1, while the CLASS_CODE 32457 might identify INFS 420, class section 2, Or, the CLASS_CODE 284458 might identify QM 362, class section 5.
- TABLE 6.5

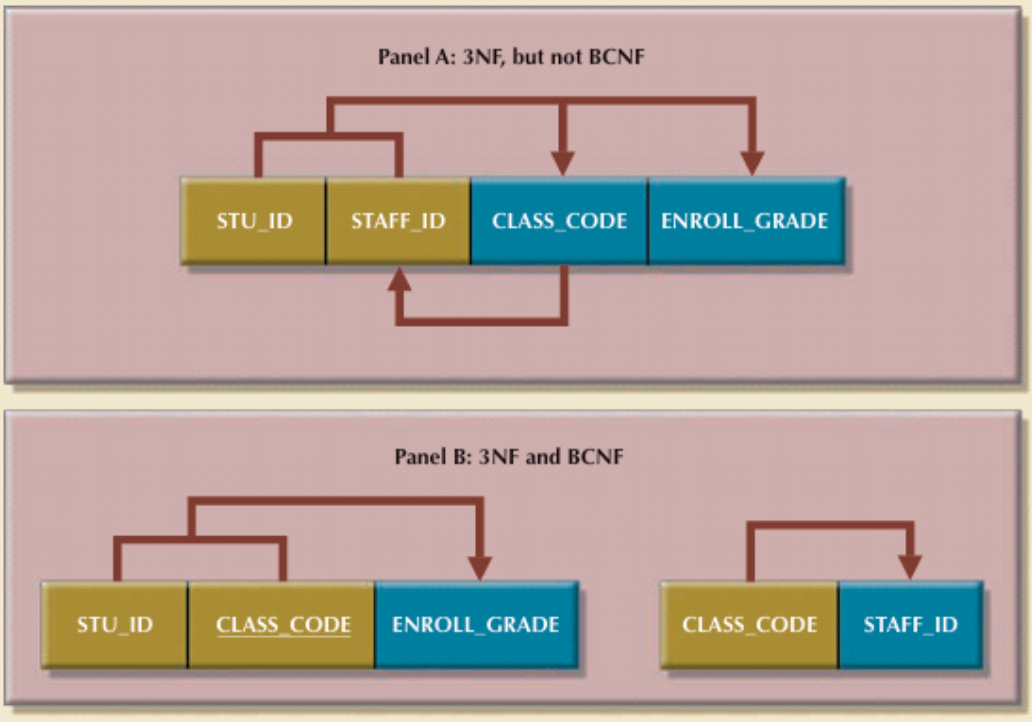
  SAMPLE DATA FOR A BCNF CONVERSION

STU_ID	STAFF_ID	CLASS_CODE	ENROLL_GRADE
125	25	21334	A
125	20	32456	C
135	20	28458	B
144	25	27563	C
144	20	32456	B

- A student can take many classes note. for example, that student 125 has taken both 21334 and 32456 earning the grade A and C, respectively.
- A staff member can teach many glasses, but each class is taught by one staff member. Note that staff member 20 teaches the class identified as 32456 and 28458

STU_ID + STAFF_ID -> CLASS_CODE, ENROLL_GRADE
CLASS_CODE -> STAFF_ID

FIGURE 6.10 ANOTHER BCNF DECOMPOSITION



Remember that a table is in BCNF when every determinant in that table is a candidate key. Therefore, when a table contains only 1 candidate key, 3NF and BCNF are equivalent.

Course

Course_id	Instructor	Room
C101	Prof. A	R1
C102	Prof. B	R3
C101	Prof. A	R3
C103	Prof. C	R1

Functional dependencies.

This relation is not in BCNF.

1. course_id → Instructor : Each course is taught by only one instructor.
2. Instructor → Room : Each instructor is assigned to only one room.

Instructor is not a candidate key.

we need to eliminate

Converting to BCNF: To bring this table into BCNF we need to eliminate the dependency Instructor → Room by decomposing the table.

We can split the course table:

We can split the course table:

Course_Id	Instructor
C101	Prof. A
C102	Prof. B
C103	Prof C

Course_Instructor table

↑

Course_id is a candidate key & no non-key attribute depends on a non-candidate key

Instructor	Room
Prof A	R1
Prof B	R2
Prof C	R3

Instructor Room table

↑

Instructor is a candidate key and no-non-key attribute depends on a non-candidate key.

This decomposition satisfies BCNF, as all functional dependencies in each table have determinants that are candidate keys.