NORMALIZATION, DATA STORAGE AND INDEXING

Design Guidelines for Relation Schemas

- Making sure that the semantics of the attributes is clear in the schema
- Reducing the redundant information in tuples
- Reducing the NULL values in tuples
- Disallowing the possibility of generating spurious tuples

1.1 Semantics of the Relational Attributes must be clear

- GUIDELINE 1: Informally, each tuple in a relation should represent one entity or relationship instance. (Applies to individual relations and their attributes).
  - Attributes of different entities (EMPLOYEEs, DEPARTMENTs, PROJECTs) should not be mixed in the same relation
  - Only foreign keys should be used to refer to other entities
  - Entity and relationship attributes should be kept apart as much as possible.
- Bottom Line: *Design a schema that can be explained easily relation by relation. The semantics of attributes should be easy to interpret.*

Figure 14.1 A simplified COMPANY relational database schema

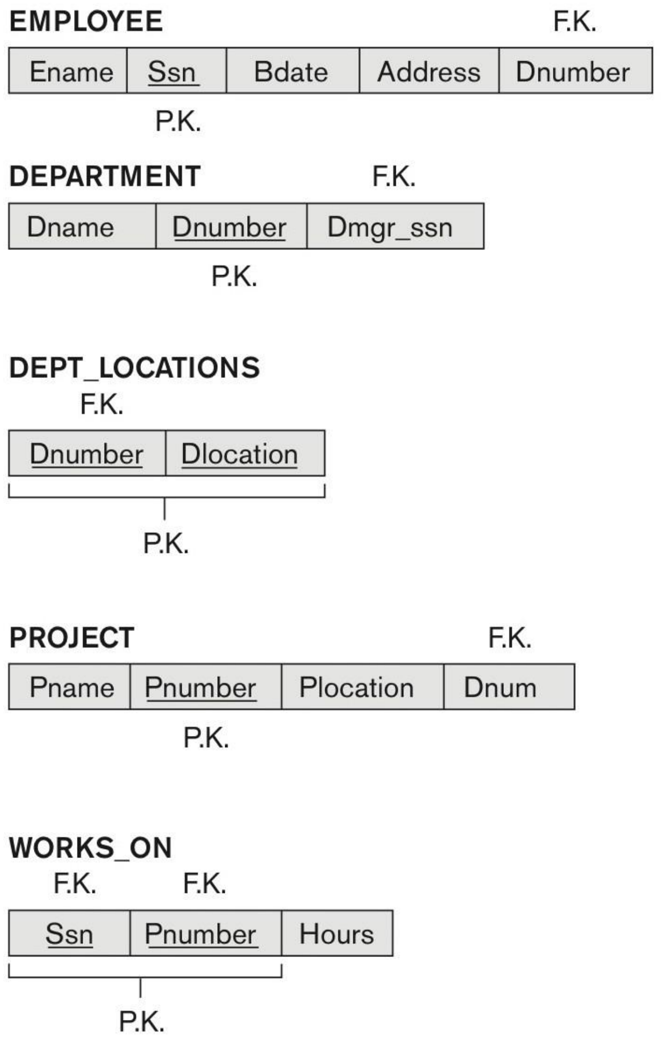


Figure 14.1 A simplified COMPANY relational database schema.

1.2 Redundant Information in Tuples and Update Anomalies

- Information is stored redundantly
  - Wastes storage
  - Causes problems with update anomalies
    - Insertion anomalies
    - Deletion anomalies
    - Modification anomalies

EXAMPLE OF AN UPDATE ANOMALY

- Consider the relation:
  - EMP_PROJ(Emp#, Proj#, Ename, Pname, No_hours)
- Update Anomaly:
  - Changing the name of project number P1 from “Billing” to “Customer-Accounting” may cause this update to be made for all 100 employees working on project P1.

EXAMPLE OF AN INSERT ANOMALY

- Consider the relation:
  - EMP_PROJ(Emp#, Proj#, Ename, Pname, No_hours)
- Insert Anomaly:
  - Cannot insert a project unless an employee is assigned to it.
- Conversely
  - Cannot insert an employee unless an he/she is assigned to a project.

EXAMPLE OF A DELETE ANOMALY

- Consider the relation:
  - EMP_PROJ(Emp#, Proj#, Ename, Pname, No_hours)
- Delete Anomaly:
  - When a project is deleted, it will result in deleting all the employees who work on that project.
  - Alternately, if an employee is the sole employee on a project, deleting that employee would result in deleting the corresponding project.

Figure 14.3 Two relation schemas suffering from update anomalies

Figure 14.3
Two relation schemas suffering from update anomalies. (a) EMP_DEPT and (b) EMP_PROJ.

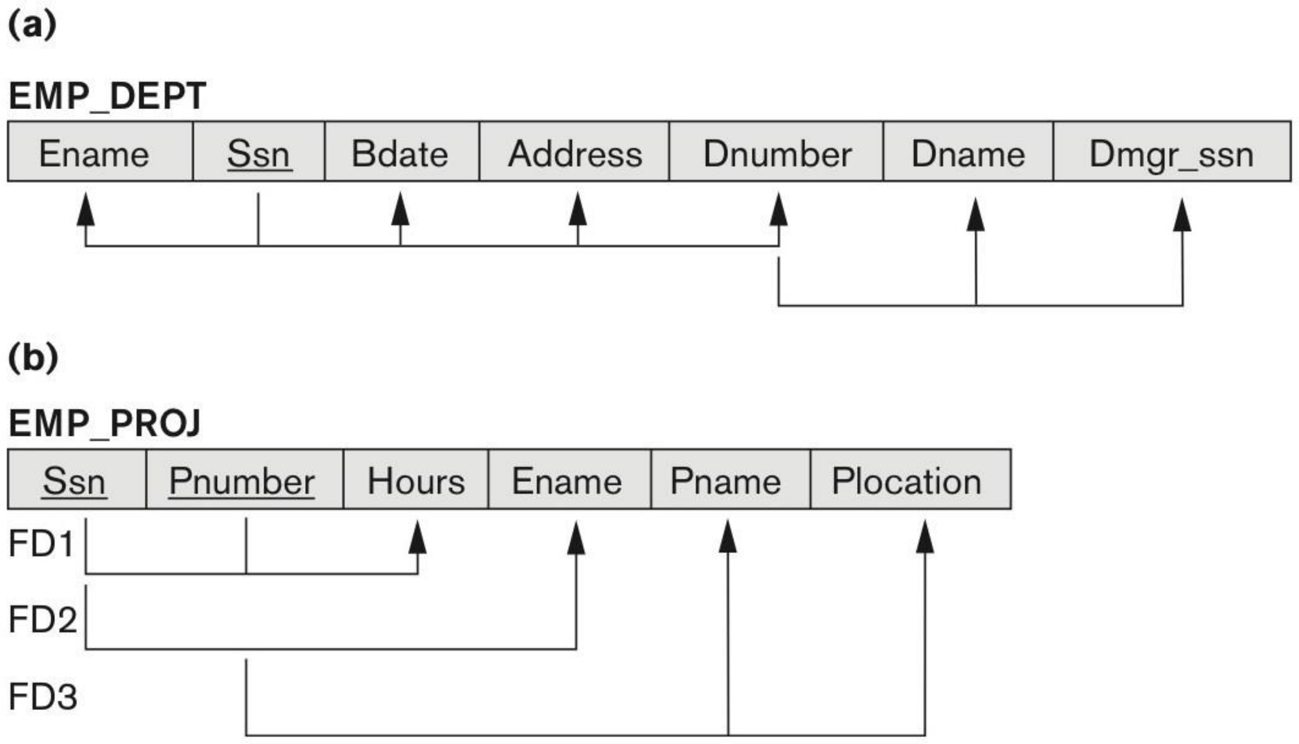
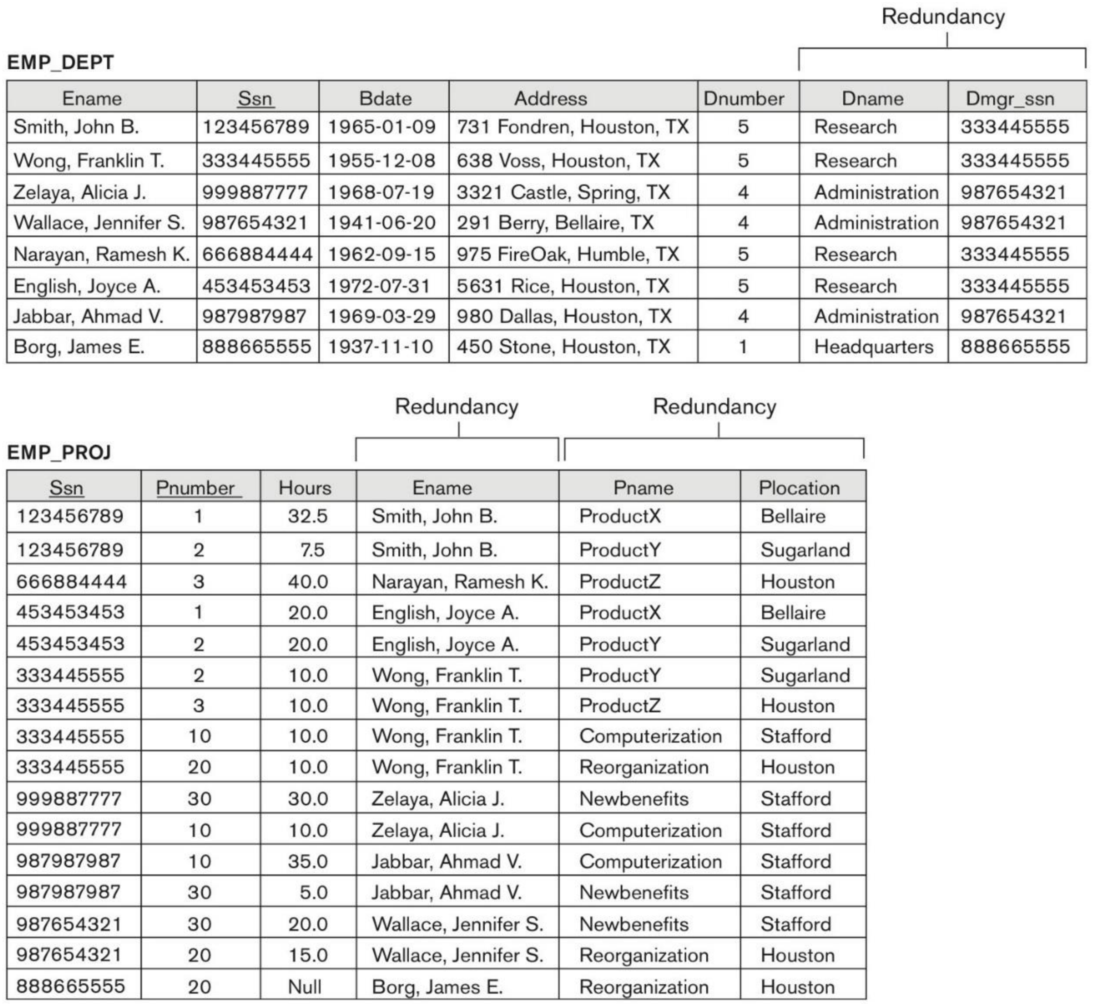


Figure 14.4 Sample states for EMP_DEPT and EMP_PROJ

Figure 14.4
Sample states for EMP_DEPT and EMP_PROJ resulting from applying NATURAL JOIN to the relations in Figure 14.2. These may be stored as base relations for performance reasons.


Redundancy
Redundancy
Redundancy

EMP_DEPT

Ename	Ssn	Bdate	Address	Dnumber	Dname	Dmgr_ssn
Smith, John B.	123456789	1965-01-09	731 Fondren, Houston, TX	5	Research	333445555
Wong, Franklin T.	333445555	1955-12-08	638 Voss, Houston, TX	5	Research	333445555
Zelaya, Alicia J.	999887777	1968-07-19	3321 Castle, Spring, TX	4	Administration	987654321
Wallace, Jennifer S.	987654321	1941-06-20	291 Berry, Bellaire, TX	4	Administration	987654321
Narayan, Ramesh K.	666884444	1962-09-15	975 FireOak, Humble, TX	5	Research	333445555
English, Joyce A.	453453453	1972-07-31	5631 Rice, Houston, TX	5	Research	333445555
Jabbar, Ahmad V.	987987987	1969-03-29	980 Dallas, Houston, TX	4	Administration	987654321
Borg, James E.	888665555	1937-11-10	450 Stone, Houston, TX	1	Headquarters	888665555

EMP_PROJ

Ssn	Pnumber	Hours	Ename	Pname	Plocation
123456789	1	32.5	Smith, John B.	ProductX	Bellaire
123456789	2	7.5	Smith, John B.	ProductY	Sugarland
666884444	3	40.0	Narayan, Ramesh K.	ProductZ	Houston
453453453	1	20.0	English, Joyce A.	ProductX	Bellaire
453453453	2	20.0	English, Joyce A.	ProductY	Sugarland
333445555	2	10.0	Wong, Franklin T.	ProductY	Sugarland
333445555	3	10.0	Wong, Franklin T.	ProductZ	Houston
333445555	10	10.0	Wong, Franklin T.	Computerization	Stafford
333445555	20	10.0	Wong, Franklin T.	Reorganization	Houston
999887777	30	30.0	Zelaya, Alicia J.	Newbenefits	Stafford
999887777	10	10.0	Zelaya, Alicia J.	Computerization	Stafford
987987987	10	35.0	Jabbar, Ahmad V.	Computerization	Stafford
987987987	30	5.0	Jabbar, Ahmad V.	Newbenefits	Stafford
987654321	30	20.0	Wallace, Jennifer S.	Newbenefits	Stafford
987654321	20	15.0	Wallace, Jennifer S.	Reorganization	Houston
888665555	20	Null	Borg, James E.	Reorganization	Houston

Guideline for Redundant Information in Tuples and Update Anomalies

- GUIDELINE 2:
  - Design a schema that does not suffer from the insertion, deletion and update anomalies.
  - If there are any anomalies present, then note them so that applications can be made to take them into account.

1.3 Null Values in Tuples

- GUIDELINE 3:
  - Relations should be designed such that their tuples will have as few NULL values as possible
  - Attributes that are NULL frequently could be placed in separate relations (with the primary key)
- Reasons for nulls:
  - Attribute not applicable or invalid
  - Attribute value unknown (may exist)
  - Value known to exist, but unavailable

1.4 Generation of Spurious Tuples – avoid at any cost

- Bad designs for a relational database may result in erroneous results for certain JOIN operations
- The "lossless join" property is used to guarantee meaningful results for join operations

- GUIDELINE 4:
  - The relations should be designed to satisfy the lossless join condition.
  - No spurious tuples should be generated by doing a natural-join of any relations.

Spurious Tuples (2)

- There are two important properties of decompositions:
  a) Non-additive or losslessness of the corresponding join
  b) Preservation of the functional dependencies.

- Note that:
  - Property (a) is extremely important and *cannot* be sacrificed.
  - Property (b) is less stringent and may be sacrificed.

Example

- Let’s say we have a relation StudentCourse

StudentID	Course	Instructor
101	DBMS	Dr. Rao
101	OS	Dr. Smith

How to Avoid Spurious Tuples?

- Use lossless decomposition (follow the lossless join property).
- Ensure that the common attribute used in decomposition is a key or part of a key.
- Follow normalization rules (especially BCNF) properly.

Functional Dependency

StudentID	CourseID	StudentName	CourseName	Grade
101	DBMS	Alice	DBMS	A
101	OS	Alice	OS	B
102	DBMS	Bob	DBMS	A

- Composite Primary Key: {StudentID, CourseID}

Functional Dependency	Reason/Explanation
StudentID → StudentName	A student has only one name
CourseID → CourseName	A course has only one name
{StudentID, CourseID} → Grade	Grade depends on the student and course combination
{StudentID, CourseID} → StudentName	Because StudentID → StudentName
{StudentID, CourseID} → CourseName	Because CourseID → CourseName

Remember

- FDs where determinant is a subset of a candidate key = Partial Dependency
- FDs where determinant is the whole candidate key = Full Dependency
- FDs where determinant determines another non-prime attribute via another = Transitive Dependency

Functional Dependencies and Candidate Keys

- A candidate key is a set of attributes that uniquely identifies each record in a table.
- Now based on this candidate key:
- If a functional dependency's left side (determinant) is the entire candidate key, it's a Full Functional Dependency.
- If the left side is only a part (subset) of the candidate key, it's a Partial Dependency.

StudentID	CourseID	StudentName	CourseName	Grade
101	DBMS	Alice	DBMS	A
101	OS	Alice	OS	B
102	DBMS	Bob	DBMS	A

- **Candidate Key = {StudentID, CourseID}**
  (Because a student can register for multiple courses and each course has many students)

1. Full Functional Dependency{StudentID, CourseID} ⟶ Grade
2. Here, both attributes together are required to determine the Grade.
3. You cannot determine the grade with just StudentID or just CourseID.
4. Hence, this is a Full Functional Dependency.

- Partial Dependency

  StudentID ⟶ StudentName
- This means you can determine a student’s name with only part of the candidate key.
- Since StudentID is only a subset of the candidate key {StudentID, CourseID}, this is a Partial Dependency.
- Same applies to:CourseID ⟶ CourseName
- Again, CourseID is only a part of the composite key.
- So, this is also a Partial Dependency.

Remember

- Partial Dependencies lead to redundancy and update anomalies.
- They are removed when moving from 1NF to 2NF during normalization.
- **Full Dependencies** are allowed in 2NF but not **transitive dependencies**, which are addressed in 3NF.

Candidate Key Vs Composite Key

Candidate Key

- A **Candidate Key** is **any minimal set of attributes** that can **uniquely identify a tuple (row)** in a relation.
- A table can have **one or more candidate keys.**
- **One of the candidate keys is chosen as the primary key**.
- It can be a **single attribute** or a **combination** of attributes.

StudentID	Email	Name
101	alice@gmail.com	Alice
102	bob@gmail.com	Bob

- StudentID is unique → Candidate KeyEmail is also unique → Candidate Key
- So, both StudentID and Email are candidate keys.

Summary

- All composite keys are candidate keys, but not all candidate keys are composite.
- Composite key = multiple attributesCandidate key = any minimal unique set (one or more attributes)

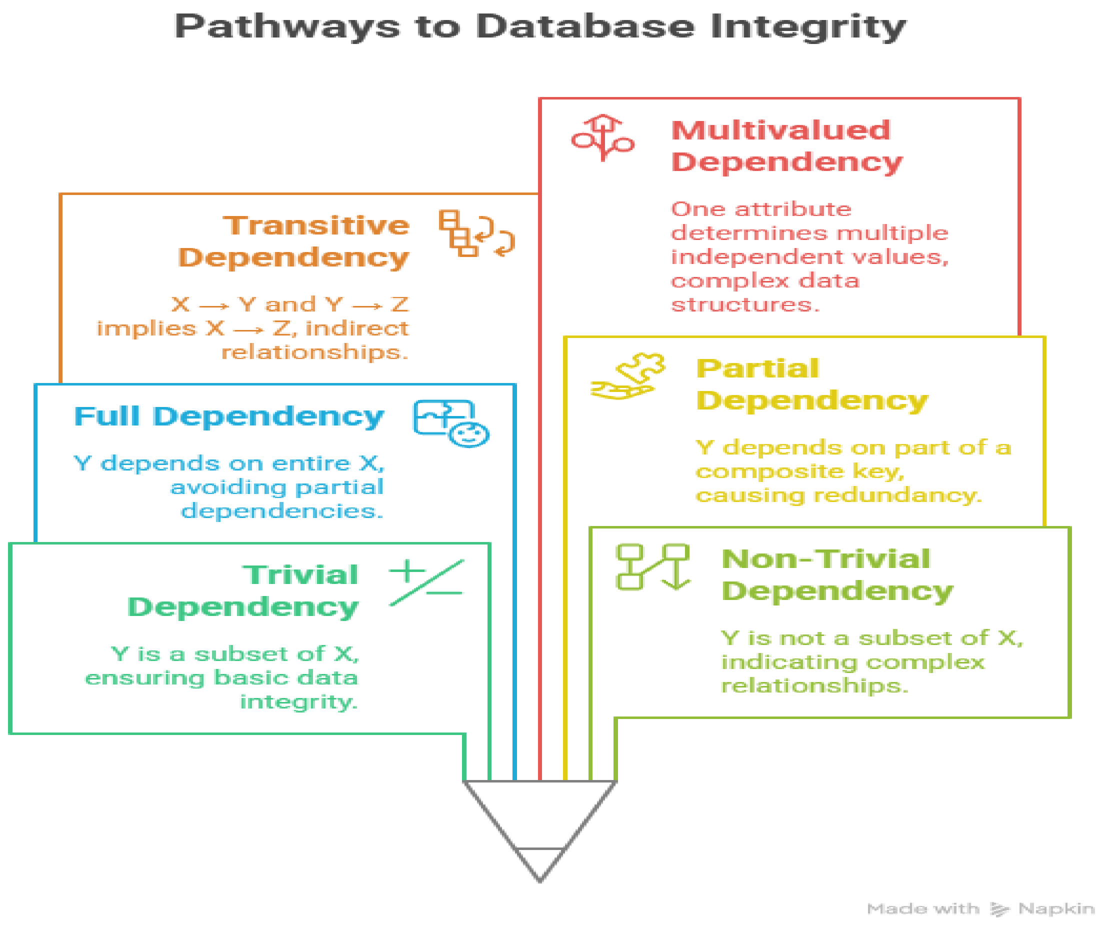
Composite Key

- A **Composite Key** is a **candidate key that is made up of more than one attribute**.
- Used when **no single attribute** can uniquely identify a row.
- So, you combine multiple columns to form a unique key.

StudentID	CourseID	Grade
101	DBMS	A
101	OS	B
102	DBMS	A

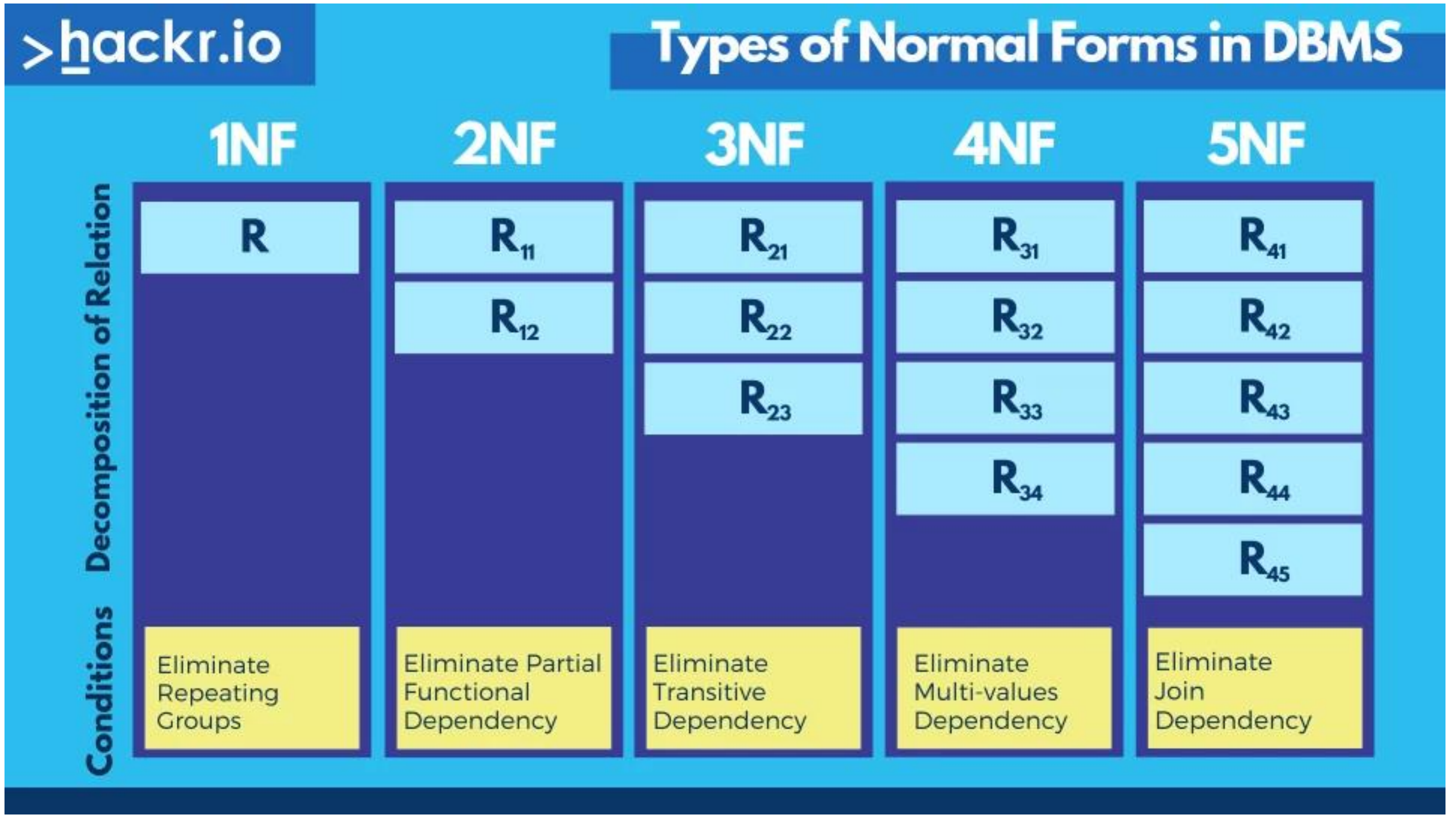
- Neither StudentID nor CourseID alone is unique.
- But the combination {StudentID, CourseID} is unique.
- {StudentID, CourseID} is a composite key (and also a candidate key).

Types of Functional dependency


Pathways to Database Integrity
Multivalued Dependency
One attribute determines multiple independent values, complex data structures.
Transitive Dependency
X ⟶ Y and Y ⟶ Z implies X ⟶ Z, indirect relationships.
Partial Dependency
Y depends on part of a composite key, causing redundancy.
Full Dependency
Y depends on entire X, avoiding partial dependencies.
Non-Trivial Dependency
Y is not a subset of X, indicating complex relationships.
Trivial Dependency
Y is a subset of X, ensuring basic data integrity.
Made with Napkin

Summary

Type	Used In	Eliminated In
Trivial FD	Always allowed	–
Non-Trivial FD	Must be managed	–
Full Functional	Required for 2NF	–
Partial Dependency	1NF	Eliminated in 2NF
Transitive Dependency	2NF	Eliminated in 3NF
Multivalued Dependency	3NF	Eliminated in 4NF
Join Dependency	4NF	Eliminated in 5NF


>hackr.io
Types of Normal Forms in DBMS
1NF
2NF
3NF
4NF
5NF
Decomposition of Relation
Conditions
R
R_{11}
R_{12}
R_{21}
R_{22}
R_{23}
R_{31}
R_{32}
R_{33}
R_{34}
R_{41}
R_{42}
R_{43}
R_{44}
R_{45}
Eliminate Repeating Groups
Eliminate Partial Functional Dependency
Eliminate Transitive Dependency
Eliminate Multi-values Dependency
Eliminate Join Dependency

First Normal Form (1NF)

- A relation will be 1NF if it contains an atomic value.
- It states that an attribute of a table cannot hold multiple values. It must hold only single-valued attribute.
- First normal form disallows the multi-valued attribute, composite attribute, and their combinations.

Example: Relation EMPLOYEE is not in 1NF because of multi-valued attribute EMP_PHONE.

EMPLOYEE table:

EMP_ID	EMP_NAME	EMP_PHONE	EMP_STATE
14	John	7272826385, 9064738238	UP
20	Harry	8574783832	Bihar
12	Sam	7390372389, 8589830302	Punjab

The decomposition of the EMPLOYEE table into 1NF has been shown below:

EMP_ID	EMP_NAME	EMP_PHONE	EMP_STATE
14	John	7272826385	UP
14	John	9064738238	UP
20	Harry	8574783832	Bihar
12	Sam	7390372389	Punjab
12	Sam	8589830302	Punjab

Issues in 1NF

- Data redundancy is higher in 1NF because there are multiple columns with the same in multiple rows.
- 1NF is not so focused on **eliminating redundancy** as much as it is focused on **eliminating repeating groups**.

Second Normal Form (2NF)

- In the 2NF, relational must be in 1NF.
- In the second normal form, all **non-key attributes** are fully functional dependent on the primary key
- This is the main feature of a primary key. The attribute that does not identify any record uniquely is called a Non-key attribute. Or the attributes that are not key or part of the key attributes are known as non-key attributes. Non-key attributes can store as many as the values repeatedly.

Example: Let's assume, a school can store the data of teachers and the subjects they teach. In a school, a teacher can teach more than one subject.

TEACHER table

TEACHER_ID	SUBJECT	TEACHER_AGE
25	Chemistry	30
25	Biology	30
47	English	35
83	Math	38
83	Computer	38

In the given table, non-prime attribute TEACHER_AGE is dependent on TEACHER_ID which is a proper subset of a candidate key. That's why it violates the rule for 2NF.

To convert the given table into 2NF, we decompose it into two tables:

TEACHER_DETAIL table:

TEACHER_ID	TEACHER_AGE
25	30
47	35
83	38

TEACHER_SUBJECT table:

TEACHER_ID	SUBJECT
25	Chemistry
25	Biology
47	English
83	Math
83	Computer

Third Normal Form (3NF)

- A relation will be in 3NF if it is in 2NF and not contain any transitive partial dependency.
- 3NF is used to reduce the data duplication. It is also used to achieve the data integrity.
- If there is no transitive dependency for non-prime attributes, then the relation must be in third normal form.

EMPLOYEE_DETAIL table:

EMP_ID	EMP_NAME	EMP_ZIP	EMP_STATE	EMP_CITY
222	Harry	201010	UP	Noida
333	Stephan	02228	US	Boston
444	Lan	60007	US	Chicago
555	Katharine	06389	UK	Norwich
666	John	462007	MP	Bhopal

Super key in the table above:

```
{EMP_ID}, {EMP_ID, EMP_NAME}, {EMP_ID, EMP_NAME, EMP_ZIP}....so on
```

Candidate key: {EMP_ID}

Non-prime attributes: In the given table, all attributes except EMP_ID are non-prime.

Here, EMP_STATE & EMP_CITY dependent on EMP_ZIP and EMP_ZIP dependent on EMP_ID. The non-prime attributes (EMP_STATE, EMP_CITY) transitively dependent on super key(EMP_ID). It violates the rule of third normal form.

That's why we need to move the EMP_CITY and EMP_STATE to the new <EMPLOYEE_ZIP> table, with EMP_ZIP as a Primary key.

EMPLOYEE table:

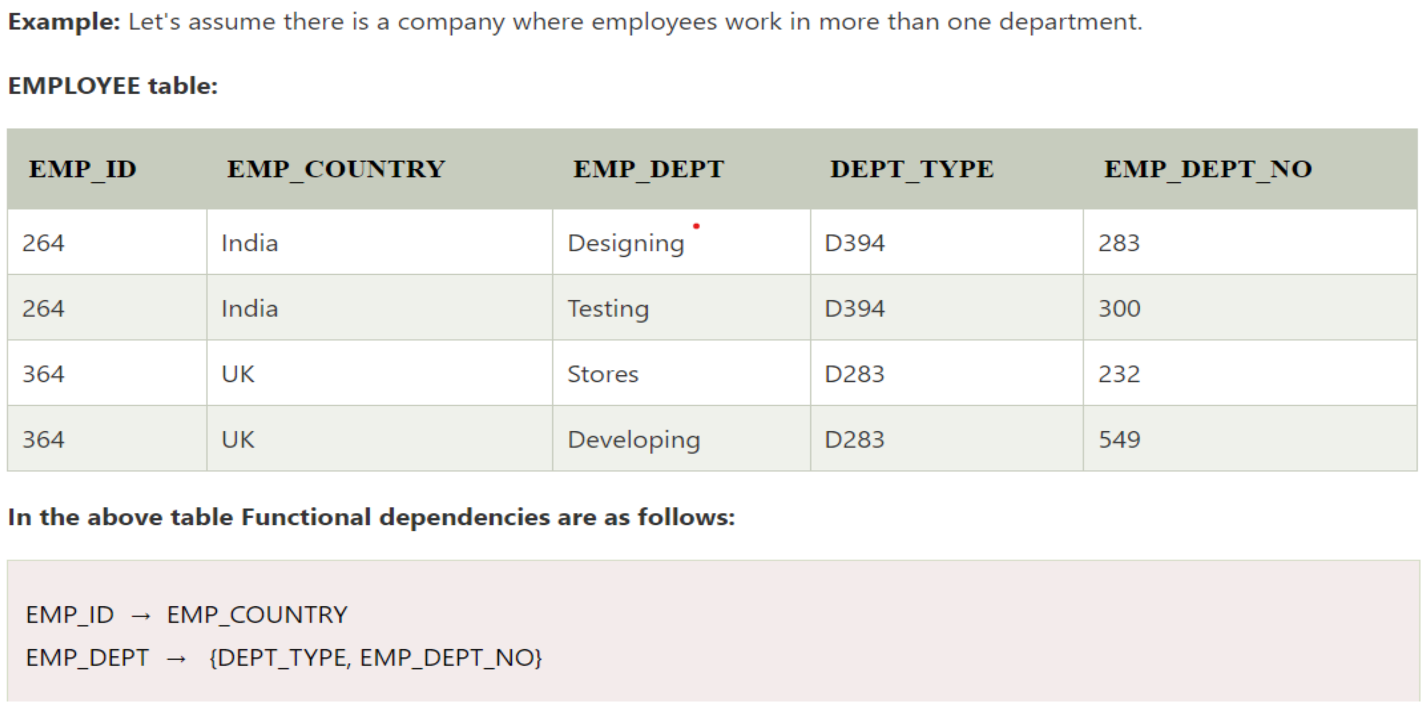
EMP_ID	EMP_NAME	EMP_ZIP
222	Harry	201010
333	Stephan	02228
444	Lan	60007
555	Katharine	06389
666	John	462007

EMPLOYEE_ZIP table:

EMP_ZIP	EMP_STATE	EMP_CITY
201010	UP	Noida
02228	US	Boston
60007	US	Chicago
06389	UK	Norwich
462007	MP	Bhopal

Boyce Codd normal form (BCNF)

- BCNF is the advance version of 3NF. It is stricter than 3NF.
- A table is in BCNF if every functional dependency X → Y, X is the super key of the table.
- For BCNF, the table should be in 3NF, and for every FD, LHS is super key.



Example: Let's assume there is a company where employees work in more than one department.

EMPLOYEE table:

EMP_ID	EMP_COUNTRY	EMP_DEPT	DEPT_TYPE	EMP_DEPT_NO
264	India	Designing	D394	283
264	India	Testing	D394	300
364	UK	Stores	D283	232
364	UK	Developing	D283	549

In the above table Functional dependencies are as follows:

EMP_ID → EMP_COUNTRY
EMP_DEPT → {DEPT_TYPE, EMP_DEPT_NO}

- **Candidate key: {EMP-ID, EMP-DEPT}**

EMP_COUNTRY table:

EMP_ID	EMP_COUNTRY
264	India
264	India

EMP_DEPT table:

EMP_DEPT	DEPT_TYPE	EMP_DEPT_NO
Designing	D394	283
Testing	D394	300
Stores	D283	232
Developing	D283	549

EMP_DEPT_MAPPING table:

EMP_ID	EMP_DEPT
D394	283
D394	300
D283	232
D283	549

Functional dependencies:

```
EMP_ID    →    EMP_COUNTRY
EMP_DEPT   →   {DEPT_TYPE, EMP_DEPT_NO}
```

Candidate keys:

For the first table: EMP_ID
For the second table: EMP_DEPT
For the third table: {EMP_ID, EMP_DEPT}

Fourth normal form (4NF)

- A relation will be in 4NF if it is in Boyce Codd normal form and has no **multi-valued dependency.**
- For a dependency A → B, if for a single value of A, multiple values of B exists, then the relation will be a multi-valued dependency.
- A table should have atleast 3 coloumns
- In a relation R(A,B,C) for a multivalued dependency A->B, then B and C should be independent of each other.

Example

STUDENT

STU_ID	COURSE	HOBBY
21	Computer	Dancing
21	Math	Singing
34	Chemistry	Dancing
74	Biology	Cricket
59	Physics	Hockey

The given STUDENT table is in 3NF, but the COURSE and HOBBY are two independent entity. Hence, there is no relationship between COURSE and HOBBY.

In the STUDENT relation, a student with STU_ID, **21** contains two courses, **Computer** and **Math** and two hobbies, **Dancing** and **Singing**. So there is a Multi-valued dependency on STU_ID, which leads to unnecessary repetition of data.

STUDENT_COURSE

STU_ID	COURSE
21	Computer
21	Math
34	Chemistry
74	Biology
59	Physics

STUDENT_HOBBY

STU_ID	HOBBY
21	Dancing
21	Singing
34	Dancing
74	Cricket
59	Hockey

Fifth normal form (5NF)

- A relation is in 5NF if it is in 4NF and not contains any join dependency and joining should be lossless.
- 5NF is satisfied when all the tables are broken into as many tables as possible in order to avoid redundancy.
- 5NF is also known as Project-join normal form (PJ/NF).

Example

SUBJECT	LECTURER	SEMESTER
Computer	Anshika	Semester 1
Computer	John	Semester 1
Math	John	Semester 1
Math	Akash	Semester 2
Chemistry	Praveen	Semester 1

In the above table, John takes both Computer and Math class for Semester 1 but he doesn't take Math class for Semester 2. In this case, combination of all these fields required to identify a valid data.

Suppose we add a new Semester as Semester 3 but do not know about the subject and who will be taking that subject so we leave Lecturer and Subject as NULL. But all three columns together acts as a primary key, so we can't leave other two columns blank.

So to make the above table into 5NF, we can decompose it into three relations P1, P2 & P3:

SEMESTER	SUBJECT
Semester 1	Computer
Semester 1	Math
Semester 1	Chemistry
Semester 2	Math

P2

SUBJECT	LECTURER
Computer	Anshika
Computer	John
Math	John
Math	Akash
Chemistry	Praveen

P3

SEMSTER	LECTURER
Semester 1	Anshika
Semester 1	John
Semester 1	John
Semester 2	Akash
Semester 1	Praveen

Functional Dependencies

- Inference Rules for FDs
- Equivalence of Sets of FDs
- Minimal Sets of FDs
- Properties of Relational Decompositions
- Nulls, Dangling Tuples, Alternative Relational Designs

1. Functional Dependencies : Inference Rules, Equivalence and Minimal Cover

- To recollect:

A set of attributes X *functionally determines* a set of attributes Y if the value of X determines a unique value for Y.

- Our goal here is to determine the properties of functional dependencies and to find out the ways of manipulating them.

Defining Functional Dependencies

- $X \longrightarrow Y$ holds if whenever two tuples have the same value for X, they *must have* the same value for Y
    - For any two tuples t1 and t2 in any relation instance r(R): If t1[X]=t2[X], *then* t1[Y]=t2[Y]
- $X \longrightarrow Y$ in R specifies a *constraint* on all relation instances r(R)
- Written as $X \longrightarrow Y$; can be displayed graphically on a relation schema (denoted by the arrow:).
- FDs are derived from the real-world constraints on the attributes

Inference Rules for FDs (1)

- **Definition:** An FD $X \rightarrow Y$ is **inferred from** or **implied by** a set of dependencies *F* specified on *R* if $X \rightarrow Y$ holds in *every* legal relation state *r* of *R*; that is, whenever *r* satisfies all the dependencies in *F*, $X \rightarrow Y$ also holds in *r*.
- Given a set of FDs F, we can **infer** additional FDs that hold whenever the FDs in F hold

Inference Rules for FDs (2)

- Armstrong's inference rules:
  - IR1. (**Reflexive**) If Y *subset-of* X, then X → Y
  - IR2. (**Augmentation**) If X → Y, then XZ → YZ
    - (Notation: XZ stands for X U Z)
  - IR3. (**Transitive**) If X → Y and Y → Z, then X → Z

- IR1, IR2, IR3 form a **sound** and **complete** set of inference rules
  - These are rules hold and all other rules that hold can be deduced from these

Inference Rules for FDs (3)

- Some additional inference rules that are useful:
  - **Decomposition:** If X ⟶ YZ, then X ⟶ Y and X ⟶ Z
  - **Union:** If X ⟶ Y and X ⟶ Z, then X ⟶ YZ
  - **Psuedotransitivity:** If X ⟶ Y and WY ⟶ Z, then WX ⟶ Z
- The last three inference rules, as well as any other inference rules, can be deduced from IR1, IR2, and IR3 (completeness property)

Why Compute Closure?

- To check if a functional dependency $X \rightarrow Y$ holds: compute X^+ and see if $Y \subseteq X^+$.
- To find candidate keys: if X^+ = all attributes in the relation, then X is a superkey (and potentially a candidate key).
- To decompose relations or check normalization.

Algorithm to determine Closure

- **Algorithm 15.1.** Determining X^+, the Closure of X under F
- **Input:** A set F of FDs on a relation schema R, and a set of attributes X, which is a subset of R.

```
$X^+ := X$;
repeat
    old$X^+ := X^+$;
    for each functional dependency $Y \rightarrow Z$ in $F$ do
        if $X^+ \supseteq Y$ then $X^+ := X^+ \cup Z$;
until ($X^+ = $ old$X^+$);
```

Example of Closure (1)

- For example, consider the following relation schema about classes held at a university in a given academic year.

CLASS (Classid, Course#, Instr_name, Credit_hrs, Text, Publisher, Classroom, Capacity).

- Let *F*, the set of functional dependencies for the above relation include the following f.d.s:

FD1: Sectionid $\rightarrow$ Course#, Instr_name, Credit_hrs, Text, Publisher, Classroom, Capacity;

FD2: Course# $\rightarrow$ Credit_hrs;

FD3: {Course#, Instr_name} $\rightarrow$ Text, Classroom;

FD4: Text $\rightarrow$ Publisher

FD5: Classroom $\rightarrow$ Capacity

These f.d.s above represent the meaning of the individual attributes and the relationship among them and defines certain rules about the classes.

Example of Closure (2)

- The closures of attributes or sets of attributes for some example sets:

{ Classid } $^+$ = { Classid , Course#, Instr_name, Credit_hrs, Text, Publisher, Classroom, Capacity } = CLASS

{ Course#} $^+$ = { Course#, Credit_hrs}

{ Course#, Instr_name } $^+$ = { Course#, Credit_hrs, Text, Publisher, Classroom, Capacity }

Note that each closure above has an interpretation that is revealing about the attribute(s) on the left-hand-side. The closure of { Classid } $^+$ is the entire relation CLASS indicating that all attributes of the relation can be determined from Classid and hence it is a key.

1.2 Equivalence of Sets of FDs

- Two sets of FDs F and G are **equivalent** if:
  - Every FD in F can be inferred from G, and
  - Every FD in G can be inferred from F
  - Hence, F and G are equivalent if $F^+ = G^+$
- Definition (**Covers**):
  - F **covers** G if every FD in G can be inferred from F
    - (i.e., if G^+ *subset-of* F^+)
- F and G are equivalent if F covers G and G covers F
- There is an algorithm for checking equivalence of sets of FDs

1.3 Finding Minimal Cover of F.D.s (1)

- Just as we applied inference rules to expand on a set *F* of FDs to arrive at *F*+, its closure, it is possible to think in the opposite direction to see if we could shrink or reduce the set *F* to its *minimal form* so that the minimal set is still equivalent to the original set *F.*
- **Definition:** An attribute in a functional dependency is considered **extraneous attribute** if we can remove it without changing the closure of the set of dependencies. Formally, given F, the set of functional dependencies and a functional dependency $X \rightarrow A$ in F, attribute Y is extraneous in X if *Y is a subset of X,* and F logically implies $(F - (X \rightarrow A) \cup \{ (X - Y) \rightarrow A \})$

Minimal Sets of FDs (2)

- A set of FDs is **minimal** if it satisfies the following conditions:
  1. Every dependency in F has a single attribute for its RHS.
  2. We cannot remove any dependency from F and have a set of dependencies that is equivalent to F.
  3. We cannot replace any dependency $X \longrightarrow A$ in F with a dependency $Y \longrightarrow A$, where Y is a proper-subset-of X and still have a set of dependencies that is equivalent to F.

Minimal Sets of FDs (3)

- **Algorithm 15.2. Finding a Minimal Cover F for a Set of Functional Dependencies E**
  - **Input: A set of functional dependencies E.**

1. Se tF:=E.
2. Replace each functional dependency X ⟶ {A1, A2, ..., An} in F by the n functional dependencies X ⟶A1, X ⟶A2, ..., X ⟶ An.
3. For each functional dependency X ⟶ A in F for each attribute B that is an element of X if { {F – {X ⟶ A} } ∪ { (X – {B}) ⟶ A} } is equivalent to F then replace X ⟶ A with (X – {B}) ⟶ A in F.
   (* The above constitutes a removal of the extraneous attribute B from X *)
4. For each remaining functional dependency X ⟶ A in F if {F – {X ⟶ A} } is equivalent to F, then remove X ⟶ A from F.
   (* The above constitutes a removal of the redundant dependency X → A from F *)

Computing the Minimal Sets of FDs (4)

We illustrate algorithm 15.2 with the following:
Let the given set of FDs be E : $\{B \longrightarrow A, D \longrightarrow A, AB \longrightarrow D\}$.We have to find the minimum cover of E.

■ All above dependencies are in canonical form; so we have completed step 1 of Algorithm 10.2 and can proceed to step 2. In step 2 we need to determine if $AB \longrightarrow D$ has any redundant attribute on the left-hand side; that is, can it be replaced by $B \longrightarrow D$ or $A \longrightarrow D$?

■ Since B ⟶ A, by augmenting with B on both sides (IR2), we have $BB \longrightarrow AB$, or $B \longrightarrow AB$ (i). However, $AB \longrightarrow D$ as given (ii).

■ Hence by the transitive rule (IR3), we get from (i) and (ii), $B \longrightarrow D$. Hence $AB \longrightarrow D$ may be replaced by $B \longrightarrow D$.

■ We now have a set equivalent to original E , say E' : $\{B \longrightarrow A, D \longrightarrow A, B \longrightarrow D\}$. No further reduction is possible in step 2 since all FDs have a single attribute on the left-hand side.

■ In step 3 we look for a redundant FD in E′. By using the transitive rule on $B \longrightarrow D$ and $D \longrightarrow A$, we derive $B \longrightarrow A$. Hence $B \longrightarrow A$ is redundant in E' and can be eliminated.

■ Hence the minimum cover of E is $\{B \longrightarrow D, D \longrightarrow A\}$.

Minimal Sets of FDs (5)

- Every set of FDs has an equivalent minimal set
- There can be several equivalent minimal sets
- There is no simple algorithm for computing a minimal set of FDs that is equivalent to a set F of FDs. The process of Algorithm 15.2 is used until no further reduction is possible.
- To synthesize a set of relations, we assume that we start with a set of dependencies that is a minimal set
    - E.g., see algorithm 15.4

To check for more CK

1. Check whether any one of prime attributes are on RHS of any FD.
2. If not, there is only one CK.
3. If yes, Replace prime attribute in CK with corresponding LHS of FD.
4. Repeat finding Super key and candidate key.

Algorithm – Minimal cover/Canonical cover

1. Write the FD in a such a way that it contains right side one attribute
2. Find the closure of attribute
3. Check whether any left side attribute can be reduced