Functional Dependencies

FD's are constraints on well-formed relations and represent a formalism on the infrastructure of relation.

Definition: A *functional dependency* (FD) on a relation schema **R** is a constraint ***X*** → ***Y*****,** where *X* and *Y* are subsets of attributes of **R.**

Definition: an FD is a relationship between an attribute "Y" and a determinant (1 or more other attributes) "X" such that for a given value of a determinant the value of the attribute is uniquely defined.

- X is a determinant
- X determines Y
- Y is functionally dependent on X
- X → Y
- X →Y is trivial if Y ⊆ X
-

Definition: An FD ***X*** → ***Y*** is *satisfied* in an instance **r** of **R** if for every pair of tuples, *t* and s: if *t* and *s* agree on all attributes in *X* then they must agree on all attributes in *Y*

A key constraint is a special kind of functional dependency: all attributes of relation occur on the right-hand side of the FD:

- *SSN → SSN, Name, Address*

Example Functional Dependencies

Let R be **NewStudent**(***stuId, lastName, major, credits, status, socSecNo***)

FDs in R include

- ***{stuId}→{lastName}***, but not the reverse
- ***{stuId} →{lastName, major, credits, status, socSecNo, stuId}***
- ***{socSecNo} →{stuId, lastName, major, credits, status, socSecNo}***
- ***{credits}→{status}*****,** but not ***{status}→{credits}***

ZipCode→AddressCity

- 16652 is Huntingdon's ZIP

ArtistName→BirthYear

- Picasso was born in 1881

Autobrand→Manufacturer, *Engine type*

- Pontiac is built by General Motors with gasoline engine

Author, Title→PublDate

- Shakespeare's Hamlet was published in 1600

Trivial Functional Dependency

The FD X→Y is ***trivial*** if set {Y} is a subset of set {X}

Examples: If A and B are attributes of R,

- {A}→{A}
- {A,B} →{A}
- {A,B} →{B}
- {A,B} →{A,B}

are all trivial FDs and will not contribute to the evaluation of normalization.

FD Axioms

Understanding: Functional Dependencies are recognized by analysis of the real world; no automation or algorithm. Finding or recognizing them are the database designer's task.

FD manipulations:

- **Soundness** -- no incorrect FD's are generated
- **Completeness** -- all FD's can be generated

Axiom Name	Axiom	Example
Reflexivity	a is set of attributes, b ⊆ a, then a →b	*SSN,Name → SSN*
Augmentation	if a→ b holds and c is a set of attributes, then ca→cb	*SSN → Name* then *SSN,Phone → Name, Phone*
Transitivity	if a →b holds and b→c holds, then a→ c holds	*SSN →Zip* and *Zip → City* then *SSN →City*
Union or Additivity *	if a → b and a → c holds then a→ bc holds	*SSN→Name* and *SSN→Zip* then *SSN→Name,Z*
Decomposition or Projectivity*	if a → bc holds then a → b and a → c holds	*SSN→Name,Zip* then *SSN→Name* and *SSN→Z*
Pseudotransitivity*	if a → b and cb → d hold then ac → d holds	*Address → Project* and *Project,Date →Amount* then *Address,Date → Amount*
(NOTE)	*ab→ c does NOT imply a → b and b → c*	

*Armstrong's Axioms (basic axioms)

Closure

Find all FD's for attributes *a* in a relation R

a^+ denotes the set of attributes that are functionally determined by *a*

IF attribute(s) a IS/ARE A SUPERKEY OF R THEN a^+ SHOULD BE THE WHOLE RELATION R. **This is our goal. Any attributes in a relation not part of the closure indicates a problem with the design.**

Algorithm for Closure

```
result := a; //start with superkey a
WHILE (more changes to result) DO
  FOREACH ( FD b → c in R) DO
    IF b ⊆ result
      THEN result := result ∪ c
```

Trivial functional dependency in DBMS with example

The dependency of an attribute on a set of attributes is known as trivial functional dependency if the set of attributes includes that attribute.

Symbolically: A ->B is trivial functional dependency if B is a subset of A.

The following dependencies are also trivial: A->A & B->B

For example: Consider a table with two columns Student_id and Student_Name.

{Student_Id, Student_Name} -> Student_Id is a trivial functional dependency as Student_Id is a subset of {Student_Id, Student_Name}. That makes sense because if we know the values of Student_Id and Student_Name then the value of Student_Id can be uniquely determined.

Also, Student_Id -> Student_Id & Student_Name -> Student_Name are trivial dependencies too.

Trivial Functional Dependency

- **Trivial** – If a functional dependency (FD) X → Y holds, where Y is a subset of X, then it is called a trivial FD. Trivial FDs always hold.
- **Non-trivial** – If an FD X → Y holds, where Y is not a subset of X, then it is called a non-trivial FD.
- **Completely non-trivial** – If an FD X → Y holds, where x intersect Y = Φ, it is said to be a completely non-trivial FD.

Functional dependency in DBMS

The attributes of a table is said to be dependent on each other when an attribute of a table uniquely identifies another attribute of the same table.

For example: Suppose we have a student table with attributes: Stu_Id, Stu_Name, Stu_Age. Here Stu_Id attribute uniquely identifies the Stu_Name attribute of student table because if we know the student id we can tell the student name associated with it. This is known as functional dependency and can be written as Stu_Id->Stu_Name or in words we can say Stu_Name is functionally dependent on Stu_Id.

Formally:
If column A of a table uniquely identifies the column B of same table then it can represented as A->B (Attribute B is functionally dependent on attribute A)

Types of Functional Dependencies

- **Trivial functional dependency**
- **non-trivial functional dependency**
- **Multivalued dependency**
- **Transitive dependency**

Trivial functional dependency

The dependency of an attribute on a set of attributes is known as trivial functional dependency if the set of attributes includes that attribute.

Symbolically: A ->B is trivial functional dependency if B is a subset of A.

The following dependencies are also trivial: A->A & B->B

For example: Consider a table with two columns Student_id and Student_Name.

{Student_Id, Student_Name} -> Student_Id is a trivial functional dependency as Student_Id is a subset of {Student_Id, Student_Name}. That makes sense because if we know the values of Student_Id and Student_Name then the value of Student_Id can be uniquely determined.

Also, Student_Id -> Student_Id & Student_Name -> Student_Name are trivial dependencies too.

Non trivial functional dependency

If a functional dependency X->Y holds true where Y is not a subset of X then this dependency is called non trivial Functional dependency.

For example:
An employee table with three attributes: emp_id, emp_name, emp_address.
The following functional dependencies are non-trivial:
emp_id -> emp_name (emp_name is not a subset of emp_id)
emp_id -> emp_address (emp_address is not a subset of emp_id)

On the other hand, the following dependencies are trivial:
{emp_id, emp_name} -> emp_name [emp_name is a subset of {emp_id, emp_name}]
Refer: **trivial functional dependency**.

Completely non trivial FD:
If a FD X->Y holds true where X intersection Y is null then this dependency is said to be completely non trivial function dependency.

Multivalued dependency

Multivalued dependency occurs when there are more than one **independent** multivalued attributes in a table.

For example: Consider a bike manufacture company, which produces two colors (Black and white) in each model every year.

bike_model	manuf_year	color
M1001	2007	Black
M1001	2007	Red
M2012	2008	Black
M2012	2008	Red
M2222	2009	Black
M2222	2009	Red

Here columns manuf_year and color are independent of each other and dependent on bike_model. In this case these two columns are said to be multivalued dependent on bike_model. These dependencies can be represented like this:

bike_model ->> manuf_year

bike_model ->> color

Transitive dependency

A functional dependency is said to be transitive if it is indirectly formed by two functional dependencies. For e.g.

X -> Z is a transitive dependency if the following three functional dependencies hold true:

- X->Y
- Y does not ->X
- Y->Z

Note: A transitive dependency can only occur in a relation of three of more attributes. This dependency helps us normalizing the database in 3NF (3rd Normal Form).

Inference Rules

Armstrong's axioms are a set of axioms (or, more precisely, inference rules) used to infer all the functional dependencies on a relational database. They were developed by William W. Armstrong.
Let R(U) be a relation scheme over the set of attributes U. We will use the letters X, Y, Z to represent any subset of and, for short, the union of two sets of attributes and by instead of the usual X U Y.

- The **Armstrong's axioms** are ***very intuitive***
- **Consider the relation:**

Employee-Department

SSN	fname	lname	DNO	DName
111-11-1111	John	Smith	5	Research
222-22-2222	Jane	Doe	4	Payroll
333-33-3333	Pete	Pan	5	Research

Examples of Armostrong axioms:

1. **Reflexivity rule**: if $Y \subseteq X$ then $X \rightarrow Y$

```
{fname, lname} ⟶ {fname}

  What it says is: if I see that same values for
{fname, lname}
  I must also see that same value for {fname} - kinda
obvious :-)
```

2. **Augmentation rule**: if $X \rightarrow Y$ then $XZ \rightarrow YZ$

```
If {SSN} ⟶ {fname}   then:  {SSN, DName} ⟶ {fname,
DName}
```

3. **Transitivity rule**: if $X \rightarrow Y$ and $Y \rightarrow Z$ then $X \rightarrow Z$

```
If:

   {SSN} ⟶ {DNO}

   {DNO} ⟶ {DName}
Then also:

   {SSN} ⟶ {DName}
```

The **Decomposition rule:**

- if $X \rightarrow YZ$ then: $X \rightarrow Y$ and $X \rightarrow Z$

Union rule:

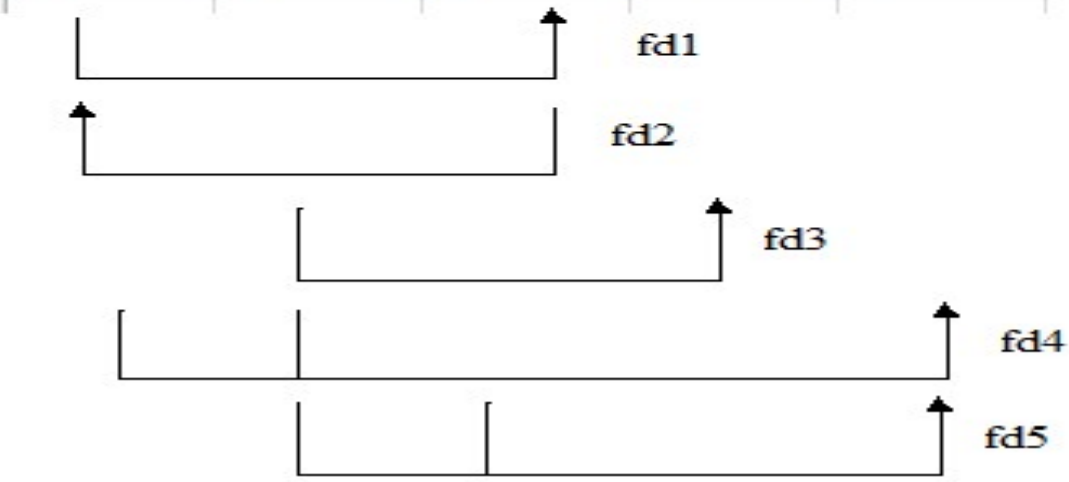
- if $X \rightarrow Y$ and $X \rightarrow Z$ then: $X \rightarrow YZ$

***Psuedo* transitivity rule**:

- if $X \rightarrow Y$ and $YW \rightarrow Z$ then: $XW \rightarrow Z$

Sample Relation:

A	B	C	D	E
a	b	z	w	q
e	b	r	w	p
a	d	z	w	t
e	d	r	w	q
a	f	z	s	t
e	f	r	s	t



A ⟶ C (fd1)
C ⟶ A (fd2)
B ⟶ D (fd3)
A,B ⟶ E (fd4)
B,C ⟶ E (fd5)

True

C, D ⟶ A (fd6)
A, B, C ⟶ E (fd7)

Not true

How to Find Candidate Key using Functional Dependencies –

In the previous post (How to Find Super Key from Functional Dependencies), we identify all the superkeys using functional dependencies. To identify Candidate Key, Let R be the relational schema, and X be the set of attributes over R. X^+ determine all the attributes of R, and therefore X is said to be superkey of R. If there are no superflous attributes in the Super key, then it will be Candidate Key.
In short, a minimal Super Key is a Candidate Key.

Example/Question 1 : Let R(ABCDE) is a relational schema with following functional dependencies. AB → C DE → B CD → E

Step 1: Identify the SuperKeys –

ACD, ABD, ADE, ABDE, ACDB, ACDE, ACDBE. {From Previous Post Eg.}

Step 2: Find minimal super key -

Neglecting the last four keys as they can be trimmed down, so, checking

the first three keys (ACD, ABD and ADE)

For SuperKey : ACD

$(A)^+$ = {A} - {Not determine all attributes of R}

$(C)^+$ = {C} - {Not determine all attributes of R}

$(D)^+$ = {D} - {Not determine all attributes of R}

For SuperKey : ABD

$(A)^+$ = {A} - {Not determine all attributes of R}

$(B)^+$ = {B} - {Not determine all attributes of R}

$(D)^+$ = {D} - {Not determine all attributes of R}

For SuperKey : ADE

$(A)^+$ = {A} - {Not determine all attributes of R}

$(D)^+$ = {D} - {Not determine all attributes of R}

$(E)^+$ = {E} - {Not determine all attributes of R}

Hence none of proper sets of SuperKeys is not able to determine all attributes of R, So ACD, ABD, ADE all are minimal superkeys or candidate keys.

Example/Question 2 : Let R(ABCDE) is a relational schema with following functional dependencies - AB → C C → D B → EA Find Out the Candidate Key ?

Step 1: Identify the super key

(AB^+) : {ABCDE} ⇒ Superkey

(C^+) : {CD} ⇒ Not a Superkey

(B^+) : {BEACD} ⇒ Superkey

So, Super Keys will be B, AB, BC, BD, BE, BAC, BAD, BAE, BCD, BCE, BDE,

BACD, BACE, BCDE, ABDE, ABCDE

Step 2: Find minimal super key -

Taking the first one key, as all other keys can be trimmed down -

(B^+) : {EABCD} {determine all the attributes of R}

Since B is a minimal SuperKey ⇒ B is a Candidate Key.

So, the Candidate Key of R is - B.

Functional Dependency Set Closure (F+)

Functional Dependency Set Closure of F is the set of all functional dependencies that are determined by it.

Example of Functional Dependency Set Closure :

Consider a relation R(ABC) having following functional dependencies :

F = { A → B, B → C }

To find the Functional Dependency Set closure of F^+ :

```
(Φ)⁺   = {Φ}

               ⇒ Φ → Φ

               ⇒ 1 FD

(A)⁺   = {ABC}

               ⇒ A → Φ,   A → A,   A → B,   A → C,

                   A → BC,  A → AB,  A → AC,  A → ABC

               ⇒ 8 FDs = (2)³

                      ... where 3 is number of attributes in closure

(B)⁺   = {BC}

               ⇒  B → Φ, B → B, B → C, B → BC
               ⇒  4 FDs = (2)²


(C)⁺   = {C}

               ⇒  C → Φ, C → C
               ⇒  2 FDs = (2)¹


(AB)⁺  = {ABC}

               ⇒  AB → Φ,  AB → A,  AB → B,  AB → C,

                   AB → AB, AB → BC, AB → AC, AB → ABC
```

⇒ 8 FDs = $(2)^3$

$(BC)^+$ = {BC}

⇒ BC → Φ, BC → B, BC → C, BC → BC
⇒ 4 FDs = $(2)^2$

$(AC)^+$ = {ABC}

⇒ AC → Φ, AC → A, AC → C, AC → C,

AC → AC, AC → AB, AC → BC, AC → ABC
⇒ 8 FDs = $(2)^3$

$(ABC)^+$ = {ABC}

⇒ ABC → Φ, ABC → A, ABC → B, ABC → C,

ABC → BC, ABC → AB, ABC → AC, ABC → ABC

⇒ 8 FDs = $(2)^3$

So, the Functional Dependency Set Closure of $(F)^+$ will be :

F^+ = {

Φ → Φ, A → Φ, A → A, A → B, A → C, A → BC, A → AB, A → AC, A → ABC,

B → Φ, B → B, B → C, B → BC, C → Φ, C → C, AB → Φ, AB → A, AB → B,

AB → C, AB → AB, AB → BC, AB → AC, AB → ABC, BC → Φ, BC → B,

BC → C, BC → BC, AC → Φ, AC → A, AC → C, AC → C, AC → AC, AC → AB,

AC → BC, AC → ABC, ABC → Φ, ABC → A, ABC → B, ABC → C, ABC → BC,

ABC → AB, ABC → AC, ABC → ABC

}

The Total FDs will be :

1 + 8 + 4 + 2 + 8 + 4 + 8 + 8 = 43 FDs

Consider another relation R(AB) having following functional dependencies :

F = { A → B, B → A }

To find the Functional Dependency Set closure of F^+ :

```
(Φ)+   = {Φ}     ⇒ 1

(A)+   = {AB}    ⇒ 4 = (2)²

(B)+   = {AB}    ⇒ 4 = (2)²
(AB)+  = {AB}    ⇒ 4 = (2)²

           Total = 13
```

DEPENDENCY PRESERVING EXAMPLES

R(ABCDEF) has following FD's F = {A→BCD, A→EF, BC→AD, BC→E, BC→F, B→F, D→E} D = {ABCD, BF, DE} check whether decomposition is dependency preserving or not

Solution :
The following dependencies can be projected into the following decomposition

ABCD (R1)	BF (R2)	DE (R3)
A → BCD BC → AD	B → F	D → E

The FDs in the table are preserved also as they are projected in their corresponding decomposition.
Check whether BC → E and A → EF preserved in the decomposition or not
Apply the algorithm above :
For BC → E :
Let Z = BC

```
(Z ∩ R1)                = BC ∩ ABCD     = BC     // Intersection
(Z ∩ R1)+               = (BC)+           = BCADEF // Closure
{(Z ∩ R1)+ ∩ R1}        = BCADEF ∩ ABCD = ABCD   // Intersection
Z= [{(Z ∩ R1)+ ∩ R1}∪Z] = ABCD ∪ BC     = ABCD  // Updating Z (Union)
```

Z does not contain E(RHS of FD- BC → E), Repeating procedure with R_2

Now, z = ABCD

```
(Z ∩ R2)                = ABCD ∩ BF  = B      // Intersection
(Z ∩ R2)+               = B+         = BF     // Closure
{(Z ∩ R2)+ ∩ R2}        = BF ∩ BF    = BF     // Intersection
Z= [{(Z ∩ R2)+ ∩ R2}∪Z] = ABCD ∪ BF = ABCDF  // Updating Z (Union)
```

Z does not contain E(RHS of FD- BC → E), Repeating procedure with R_3

Now Z = ABCDF
$(Z \cap R_3)$ = ABCDF ∩ DE = B // Intersection
$(Z \cap R_3)^+$ = D^+ = DE // Closure
$\{(Z \cap R_3)^+ \cap R_3\}$ = DE ∩ DE = BF // Intersection
$Z= [\{(Z \cap R_3)^+ \cap R_3\} \cup Z]$ = DE ∪ ABCDF = ABCDEF // Updating Z (Union)

Stopping the algorithm as Z contains E, So BC→E preserves dependency

For A → EF :
Let Z = A
$(Z \cap R_1)$ = A ∩ ABCD = A // Intersection
$(Z \cap R_1)^+$ = A^+ = ABCDEF // Closure
$\{(Z \cap R_1)^+ \cap R_1\}$ = ABCDEF ∩ ABCD = ABCD // Intersection
$Z= [\{(Z \cap R_1)^+ \cap R_1\} \cup Z]$ = ABCD ∪ A = ABCD // Updating Z (Union)

Z does not contain E and F(RHS of FD- A→EF), Repeating procedure with R2

Now, z = ABCD
$(Z \cap R_2)$ = ABCD ∩ BF = B // Intersection
$(Z \cap R_2)^+$ = B^+ = BF // Closure
$\{(Z \cap R_2)^+ \cap R_2\}$ = BF ∩ BF = BF // Intersection
$Z= [\{(Z \cap R_2)^+ \cap R_2\} \cup Z]$ = ABCD ∪ BF = ABCDF // Updating Z (Union)

A→F preserves dependency. Checking for A→E, Repeating procedure with R3

Now Z = ABCDF
$(Z \cap R_3)$ = ABCDF ∩ DE = B // Intersection
$(Z \cap R_3)^+$ = D^+ = DE // Closure
$\{(Z \cap R_3)^+ \cap R_3\}$ = DE ∩ DE = BF // Intersection
$Z= [\{(Z \cap R_3)^+ \cap R_3\} \cup Z]$ = DE ∪ ABCDF = ABCDEF // Updating Z (Union)

Stopping the algorithm as Z contains F, So A→F also preserves dependency

Question 1: R(ABCD) F = {A → B, B → C, C → D, D → A} D = {AB, BC, CD} Check whether the decomposition is preserving dependency or not ?

Solution :
The following dependencies can be projected into the following decomposition :

R1(AB)	R2(BC)	R3(CD)

A→B	B→C	C → D

Inferring reverse FDs which fits into the decomposition-
B+ w.r.t F = {BCDA} ⇒ B → A
C+ w.r.t F = {CDAB} ⇒ C → B
D+ w.r.t F = {DABC} ⇒ D → C

So, the table will be updated as -

R1(AB)	R2(BC)	R3(CD)
A→B B→A	B→C C→B	C → D D→C

Checking D → A preserves dependency or not -
Compute D+ w.r.t updated table FDs :
D+ = {DCBA}
as closure of D w.r.t updated table FDs contains A. So D→A preserves dependency.

Question 3: R(ABCDEG) F = {AB → C, AC → B, BC → A, AD → E, B → D, E → G} D = {ABC, ACDE, ADG} Check whether the decomposition is preserving dependency or not ?

Solution :
The following dependencies can be projected into the following decomposition :

R1(ABC)	R2(ACDE)	R3(ADG)
AB→C AC→B BC→A	AD→E	

Inferring reverse FDs which fits into the decomposition-
C+ w.r.t F = {C}
B+ w.r.t F = {BD}
A+ w.r.t F = {A}
E+ w.r.t F = {EG}
No reverse FDs can be derived.

Checking B → D preserves dependency or not -
Compute B+ w.r.t updated table FDs :
B+ = {B}
as closure of B w.r.t table FDs doesn't contains D. So B→D doesn't preserves dependency.

Checking E → G preserves dependency or not -
Compute E+ w.r.t updated table FDs :
E+ = {E}
as closure of E w.r.t table FDs doesn't contains G. So E→G doesn't preserves dependency.

Question 4: Let R(ABCD) be a relational schema with the following functional dependencies : F = {A → B, B → C, C → D, D → B}. The decomposition of R into D = {AB, BC, BD} Check whether the decomposition is preserving dependency or not ?

Solution :
The following dependencies can be projected into the following decomposition :

R1(AB)	R2(BC)	R3(BD)
A→B	B→C	D → B

Inferring reverse FDs which fits into the decomposition-
B+ w.r.t F = {BCD} ⇒ B → D
C+ w.r.t F = {CDB} ⇒ C → B

So, the table will be updated as -

R1(AB)	R2(BC)	R3(CD)
A→B	B→C C→B	D→B B→D

Checking C → D preserves dependency or not -
Compute C+ w.r.t updated table FDs :
C+ = {CBD}
as closure of C w.r.t updated table FDs contains D. So C→D preserves dependency.

Question 5: R(ABCDE) F = {A → BC, CD → E, B → D, E → A} D = {ABCE, BD} Check whether the decomposition is preserving dependency or not ?

Solution :
The following dependencies can be projected into the following decomposition :

R1(ABCE)	R2(BD)
A→B A→C E→A	B→D

Inferring reverse FDs which fits into the decomposition-
B+ w.r.t F = {BD}
C+ w.r.t F = {C}
A+ w.r.t F = {ABCDE} ⇒ A → E
D+ w.r.t F = {D}

So, the table will be updated as -

R1(AB)	**R2(BC)**
A→B **A→C** **E→A** **A→E**	**B→D**

<u>Checking CD → E preserves dependency or not</u> -
Compute CD+ w.r.t updated table FDs :
CD+ = {CD}
as closure of D w.r.t updated table FDs doesn't contains E. So CD→E doesn't preserves dependency.