# Functional Dependency

If R is a relation with attributes X and Y, a functional dependency between the attributes is represented as X->Y, which specifies Y is functionally dependent on X. Here X is a determinant set and Y is a dependent attribute. Each value of X is associated with precisely one Y value.

**Closure of set of functional dependency**

Closure of set of functional dependencies F is set of all FDs that include F as well as all dependencies that can be inferred from F.

Denoted as – F+

Inference rule

Armstrong’s axioms are a set of inference rules used to infer all the functional dependencies on a relational database.

**Axiom of reflexivity**

This axiom says, if Y is a subset of X, then X determines Y



### Axiom of augmentation

The axiom of augmentation, also known as a partial dependency, says if X determines Y, then XZ determines YZ for any Z



### Axiom of transitivity

The axiom of transitivity says if X determines Y, and Y determines Z, then X must also determine Z



### Union

This rule suggests that if two tables are separate, and the PK is the same, you may want to consider putting them together. It states that if X determines Y and X determines Z then X must also determine Y and Z



For example, if:

* SIN —> EmpName
* SIN —> SpouseName

You may want to join these two tables into one as follows:

SIN –> EmpName, SpouseName

### Decomposition

Decomposition is the reverse of the Union rule. If you have a table that appears to contain two entities that are determined by the same PK, consider breaking them up into two tables. This rule states that if X determines Y and Z, then X determines Y and X determines Z separately



## Dependency Diagram

A dependency diagram, shown in Figure 11.6, illustrates the various dependencies that might exist in a non-normalized table. A non-normalized table is one that has data redundancy in it.



The following dependencies are identified in this table:

* ProjectNo and EmpNo, combined, are the PK.
* Partial Dependencies:
	+ ProjectNo —> ProjName
	+ EmpNo —> EmpName, DeptNo,
	+ ProjectNo, EmpNo —> HrsWork
* Transitive Dependency:
	+ DeptNo —> DeptName

**Closure of attributes[X-closure]**

Closure of attributes(x+) is set of attributes which can be determined using X.

Given a set α of attributes of R and a set of functional dependencies F, we need a way to find all of the attributes of R that are functionally determined by α. This set of attributes is called the **closure of** α **under F** and is denoted α+. Finding α+ is useful because:

* if α+ = R, then α is a superkey for R
* if we find α+ for all α⊆ R, we've computed F+ (except that we'd need to use decomposition to get all of it).

***Problem:***

Compute the closure for relational schema
R={A,B,C,D,E}
A-->BC
CD-->E
B-->D
E-->A
List candidate keys of R.

***Solution:***

R={A,B,C,D,E}

F, the set of functional dependencies **A-->BC, CD-->E, B-->D, E-->A**

Compute the closure for each β in β → γ in F

**Closure for A**

|  |  |  |
| --- | --- | --- |
| **Iteration** | **Result** | **using** |
| 1 | A |  |
| 2 | ABC | A-->BC |
| 3 | ABCD | B-->D |
| 4 | ABCDE | CD-->E |
| 5 | ABCDE |  |

A+ = ABCDE, Hence A is a super key

**Closure for CD**

|  |  |  |
| --- | --- | --- |
| **Iteration** | **Result** | **using** |
| 1 | CD |  |
| 2 | CDE | CD-->E |
| 3 | ACDE | E-->A |
| 4 | ABCDE | A-->BC |
| 5 | ABCDE |  |

CD+ = ABCDE, Hence CD is a super key

**Closure for B**

|  |  |  |
| --- | --- | --- |
| **Iteration** | **result** | **Using** |
| 1 | B |  |
| 2 | BD | B-->D |
| 3 | BD |  |

B+ = BD, Hence B is NOT a super key

Try applying Armstrong axioms, to find alternate keys.

B-->D

BC-->CD (by Armstrong’s augmentation rule)

**Closure for BC**

|  |  |  |
| --- | --- | --- |
| **Iteration** | **result** | **using** |
| 1 | BC |  |
| 2 | BCD | BC-->CD |
| 3 | BCDE | CD-->E |
| 4 | ABCDE | E-->A |

BC+ = ABCDE, , Hence BC is a super key

**Closure for E**

|  |  |  |
| --- | --- | --- |
| **Iteration** | **result** | **using** |
| 1 | E |  |
| 2 | AE | E-->A |
| 3 | ABCE | A-->BC |
| 4 | ABCDE | B-->D |
| 5 | ABCDE |  |

E+ = ABCDE

A and E are minimal super keys.

To see whether CD is a minimal super key, check whether its subsets are super keys.

C+ = C

D+ = D

Since C and D are not super keys, CD is a minimal super key.

To see whether BC is a minimal super key, check whether its subsets are super keys.

B+ = BD

C+ = C

Since B and C are not super keys, BC is a minimal super key.

**Since A, BC, CD, E are minimal super keys, they are the candidate keys.**A, BC, CD, E

If there are 5 attributes, then we need to check 32 (25 )combinations to find all super keys. Since we are interested only in the candidate keys, the best bet is to check closure of attributes in the left hand side of functional dependencies.

If the closure yields the relation R, it is super key. Check whether it is a minimal super key, by checking closure for its subsets.

If the closure didn’t yield the relation R, it is not a super key. Try applying Armstrong’s axioms, to get an attribute combination that is a super key. Check to see it also an minimal super key.

The list of minimal super keys obtained is the candidate keys for that relation.

A **superkey** is a set of one or more attributes that allow entities (or relationships) to be uniquely identified.

Examples for the given problem:

A,CD, E, BC, AE, AB, ABE,ACD,BCD, DE etc.

(Any attribute added with the minimal super keys A, CD, E is also a super key).

A **candidate key** is a superkey that has no superkeys as proper subsets. A candidate key is a minimal superkey.

Examples for the given problem:

A, BC, CD, E

The **primary key** is the (one) candidate key chosen (by the database designer or database administrator) as the primary means of uniquely identifying entities (or relationships).

Example for the given problem:

Any one of the above three (A, BC, CD, E) chosen by the database designer.









