Functional Dependency

A functional dependency is defined as a constraint between two sets of attributes in a relation from a database.

Given a relation *R*, a set of attributes *X* in *R* is said to **functionally determine** another attribute *Y*, also in *R*, (written $X \rightarrow Y$) if and only if each *X* value is associated with at most one *Y* value.

In other words….

X is the *determinant set* and *Y* is the *dependent attribute*. Thus, given a tuple and the values of the attributes in *X*, one can determine the corresponding value of the *Y* attribute.

Formal Definition Functional Dependencies

- A set of attributes X *functionally determines* a set of attributes Y if the value of X determines a unique value for Y
- X →Y holds if whenever two tuples have the same value for X, they *must have* the same value for Y

  If t1[X]=t2[X], *then* t1[Y]=t2[Y] in any relation instance r(R)
- X → Y in R specifies a *constraint* on all relation instances r(R)
- FDs are derived from the real-world constraints on the attributes

Determinant

Functional Dependency

EmpNum → EmpEmail

Attribute on the LHS is known as the ***determinant***

- EmpNum is a determinant of EmpEmail

Motivation for Functional Dependencies

- Functional depencies are important in database design because they allow us to eliminate redundancies.
- Using functional dependencies, you can apply Database normalization - and thereby create a more efficient database.
- Detect when a relation has redundant information.

Example

Employee

SSN	Name	JobType	DeptName
557-78-6587	Smith Jon	Accountant	Accounts
214-45-2398	Smith Jon	Engineer	Product

Note: Name is functionally dependent on SSN because an employee's name can be uniquely determined from their SSN. Name does not determine SSN, because more than one employee can have the same name..

Types of FDs

- Trivial and Non trivial
- **Trivial Functional Dependency –**
- Some functional dependencies are said to be trivial because they are satisfied by all relations. Functional dependency of the form **A–>B is trivial if B ⊆ A**
- A trivial Functional Dependency is the one where RHS is a subset of LHS.

- Example, A-->A is satisfied by all relations involving attribute A.
- SSN-->SSN
- PNUMBER-->PNUMBER, Pname
- SSN, PNUMBER -->PNUMBER
- PNUMBER --> PNUMBER, Address

Non trivial FDs

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

On the other hand, the following dependencies are trivial:

{emp_id, emp_name} -> emp_name

Closure

Let a relation *R* have some functional dependencies *F* specified. The *closure of F* (usually written as ***F+***) is the set of all functional dependencies that may be logically derived from *F*. Often *F* is the set of most obvious and important functional dependencies and ***F+***, the closure, is the set of all the functional dependencies including *F* and those that can be deduced from *F*. The closure is important and may, for example, be needed in finding one or more candidate keys of the relation.

Example

Student

SNo	SName	CNo	CName	Addr	Instr.	Office
5425	Susan	102	Maths	Pune	P. Smith	Room 112
7845	Martin	541	Bio	Mumbai	L. Talip	Room 210

SNo -> SName
SNo -> Addr

CNo -> Cname
CNo -> Instr

Instr -> Office

Too Many FDs

Using the first rule alone, from our example we have 2^7 = 128 subsets. This will further lead to many more functional dependencies. This defeats the purpose of normalizing relations.

So what now?

One way is to deal with one attribute or a set of attributes at a time and find its closure (i.e. all functional dependencies relating to them). The aim of this exercise is to find what attributes depend on a given set of attributes and therefore ought to be together.

Examples of FD constraints

- Social Security Number determines employee name

  SSN → ENAME
- Project Number determines project name and location

  PNUMBER → {PNAME, PLOCATION}
- Employee SSN and project number determines the hours per week that the employee works on the project

  {SSN, PNUMBER} → HOURS

Inference Rules for FDs

- Given a set of FDs F, we can *infer* additional FDs that hold whenever the FDs in F hold
- Armstrong's inference rules
  - A1. (Reflexive) If Y subset-of X, then X → Y
  - A2. (Augmentation) If X → Y, then XZ → YZ
    (Notation: XZ stands for X ∪ Z)
  - A3. (Transitive) If X → Y and Y → Z, then X → Z
- A1, A2, A3 form a *sound* and *complete* set of inference rules

Functional Dependencies

EmpNum	EmpEmail	EmpFname	EmpLname
123	jdoe@abc.com	John	Doe
456	psmith@abc.com	Peter	Smith
555	alee1@abc.com	Alan	Lee
633	pdoe@abc.com	Peter	Doe
787	alee2@abc.com	Alan	Lee

If EmpNum is the PK then the FDs:

EmpNum → EmpEmail

EmpNum → EmpFname

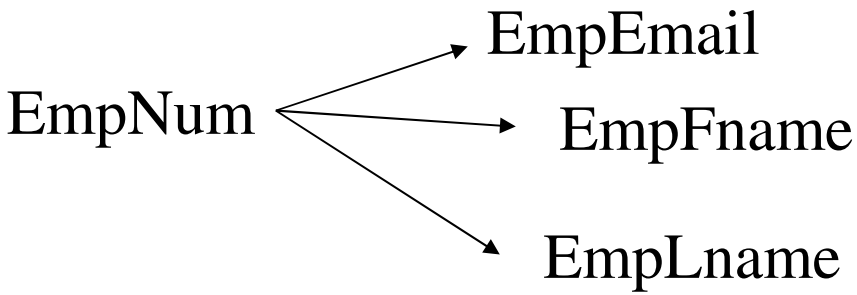
EmpNum → EmpLname

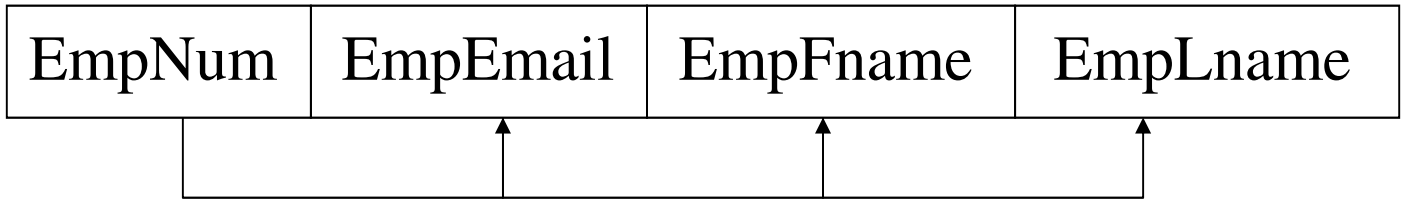
must exist.

Functional Dependencies

EmpNum → EmpEmail
EmpNum → EmpFname
EmpNum → EmpLname

3 different ways you might see FDs depicted





Transitive dependency

Transitive dependency

Consider attributes A, B, and C, and where

A → B and B → C.

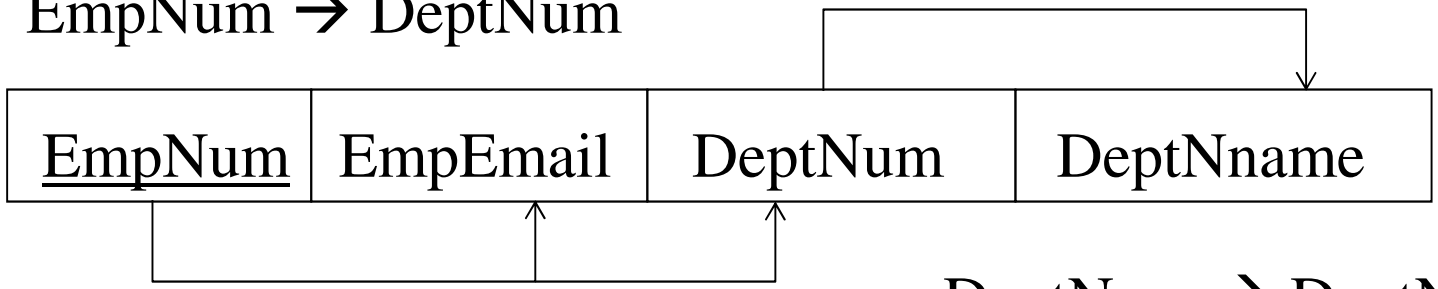
Functional dependencies are transitive, which means that we also have the functional dependency

A → C

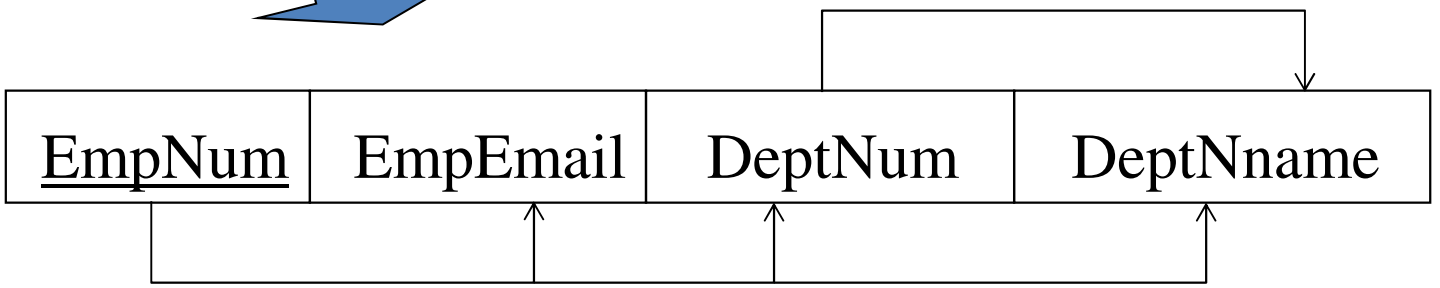
We say that C is transitively dependent on A through B.

Transitive dependency

EmpNum → DeptNum



DeptNum → DeptName

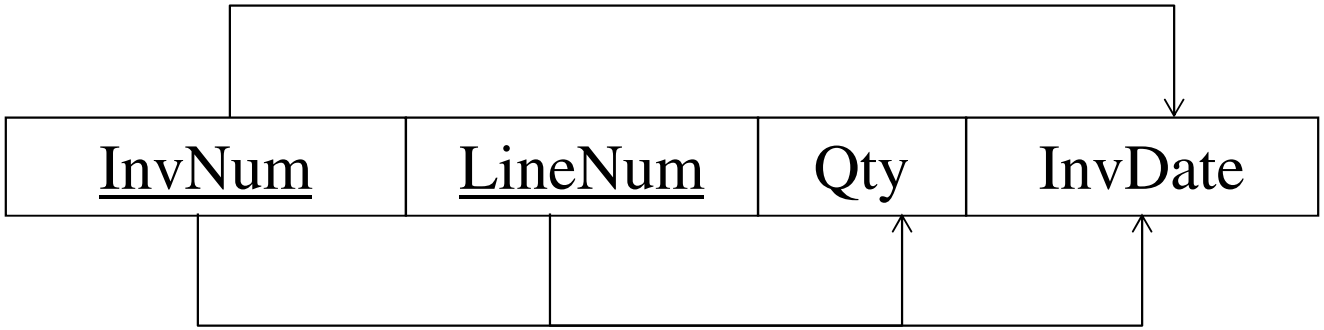


DeptName is *transitively dependent* on EmpNum via DeptNum

EmpNum → DeptName

Partial dependency

A **partial dependency** exists when an attribute B is functionally dependent on an attribute A, and A is a component of a multipart candidate key.



Candidate keys: {InvNum, LineNum} InvDate is *partially dependent* on {InvNum, LineNum} as InvNum is a determinant of InvDate and InvNum is part of a candidate key

Normalization

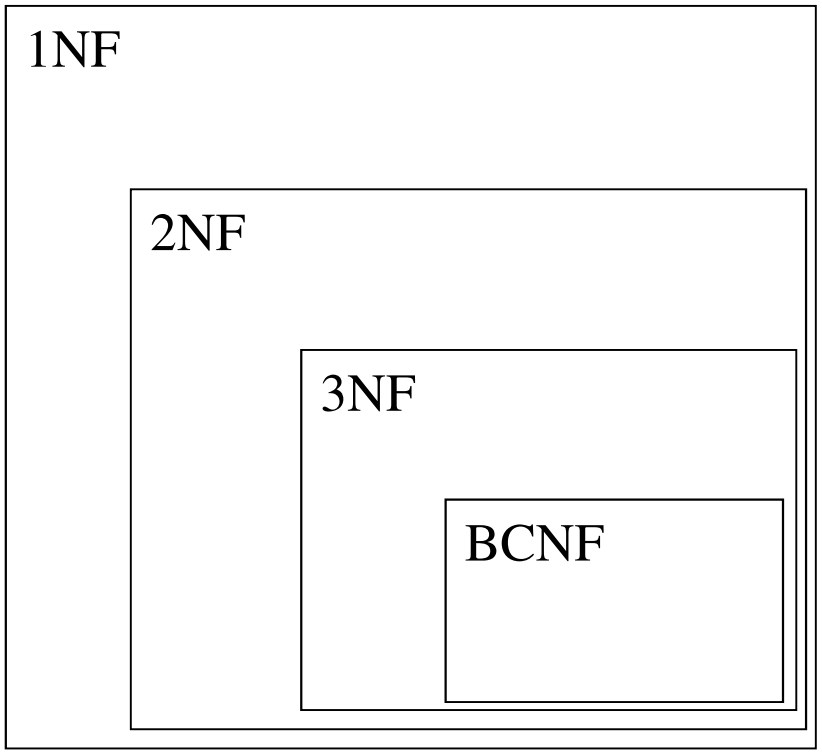
There is a sequence to normal forms:

1NF is considered the weakest,
2NF is stronger than 1NF,
3NF is stronger than 2NF, and
BCNF is considered the strongest

Also,

any relation that is in BCNF, is in 3NF;
any relation in 3NF is in 2NF; and
any relation in 2NF is in 1NF.

Normalization



a relation in BCNF, is also in 3NF

a relation in 3NF is also in 2NF

a relation in 2NF is also in 1NF

Normalization

We consider a relation in BCNF to be fully normalized.

The benefit of higher normal forms is that update semantics for the affected data are simplified.

This means that applications required to maintain the database are simpler.

A design that has a lower normal form than another design has more redundancy. Uncontrolled redundancy can lead to data integrity problems.

First we introduce the concept of ***functional dependency***

First Normal Form

First Normal Form

We say a relation is in **1NF** if all values stored in the relation are single-valued and atomic.

1NF places restrictions on the structure of relations. Values must be simple.

Normalization

Normalization

We discuss four normal forms: first, second, third, and Boyce-Codd normal forms
1NF, 2NF, 3NF, and BCNF

Normalization is a process that "improves" a database design by generating relations that are of higher normal forms.

The *objective* of normalization:
"*to create relations where every dependency is on the key, the whole key, and nothing but the key*".