CSCC43H: Introduction to Databases

Lecture 8

Wael Aboulsaadat

Acknowledgment: these slides are partially based on Prof. Garcia-Molina & Prof. Ullman slides accompanying the course's textbook.

Database Management System (DBMS)

- A collection of programs that enable:

  Defining (describing the structure),

  Populating by data (Constructing),

  Manipulating (querying, updating),

  → Preserving consistency,

  – Protecting from misuse,

  – Recovering from failure, and

  – Concurrent using

  of a database.

Normalization

Normalization

- **A technique for producing a set of relations with desirable properties, given the data requirements of the applications**

- **Outline**
  - data redundancy and anomalies
  - spurious information
  - functional dependencies
  - normalisation
    - first normal form
    - second normal form
    - third normal form
    - Boyce-Codd normal form
    - fourth normal form
    - fifth normal form

The Evils of Redundancy

- *Redundancy* is at the root of several problems associated with relational schemas:
  - ***redundant storage, insert/delete/update anomalies***

- Integrity constraints, in particular *functional dependencies*, can be used to identify schemas with such problems and to suggest refinements.

- Main refinement technique: *decomposition* (replacing ABCD with, say, AB and BCD, or ACD and ABD).

Example: A Bad Relational Design

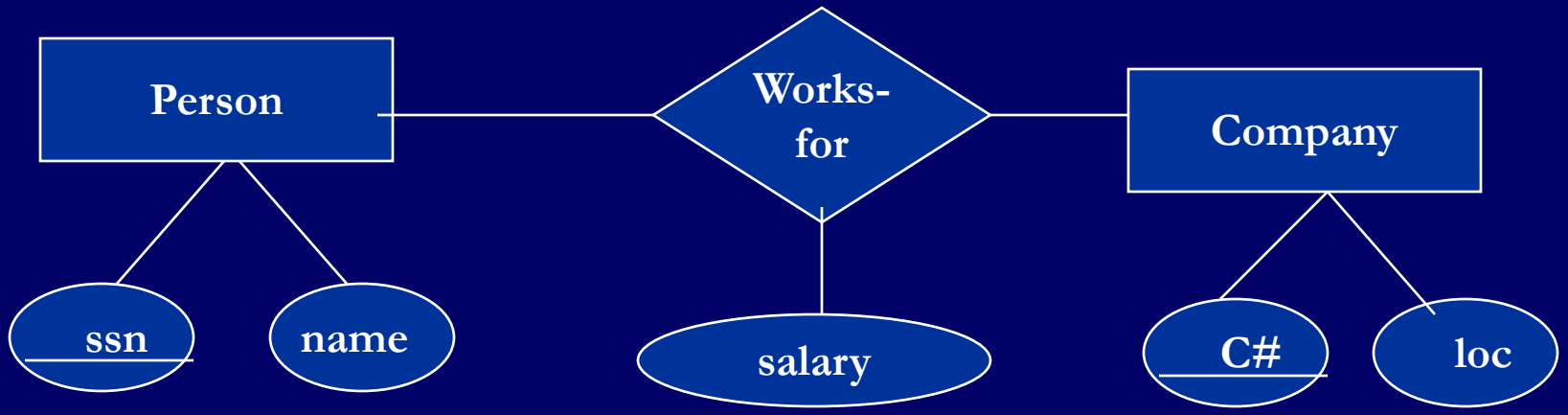


Table: X (ssn, name, salary, C#, loc)

1) Insertion Anomaly: Can we insert a person if they are not working for a company
2) Deletion Anomaly: If we delete the last employment of a company we lose the information where the company is located
3) Update Anomaly: If we change the city where a company is located we have to update multiple tuples!

Data redundancy and anomalies

Beware of keeping **multiple versions** of information.

Emp-Dept

NI#	Name	DateOfBirth	Dept#	Dname	Manager
21	AA	-	5	CS	91
22	BB	-	5	CS	91
23	CC	-	6	TS	93
24	DD	-	7	PSV	94
25	EE	-	7	PSV	94

- **Insertion** a) How do we insert a new department with no employees yet? (keys?)
  b) Entering employees is difficult as department information must be entered correctly.
- **Deletion** What happens when we delete CC's data - do we lose department 6!
- **Modification** If we change the manager of department 5, we must change it for tuples with Dept# = 5.

Spurious information

Avoid breaking up relations in such a way that **spurious information** is created

PROJECT

NI#	Name	ProjName	ProjLocation
123	XX	Accounts	London
123	XX	Analysis	Paris
124	YY	PI	London

may be broken into:

NI#	Name	ProjName
123	XX	Accounts
123	XX	Analysis
124	YY	PI

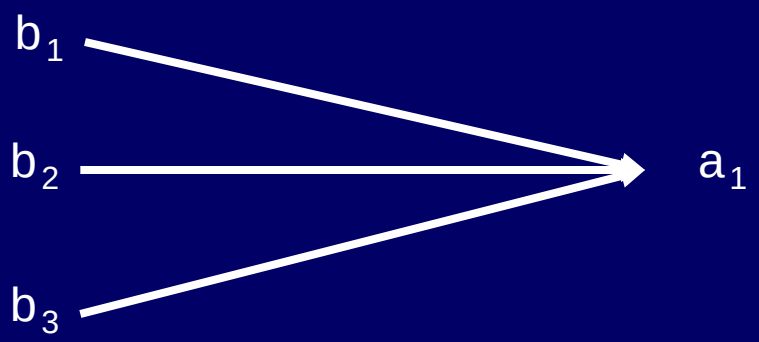
NI#	ProjLocation
123	London
123	Paris
124	London

Joining them back together, we get NEW TUPLES!

NI#	Name	ProjName	ProjLocation
123	XX	Accounts	Paris
123	XX	Analysis	London

Functional Dependence

An attribute A is functionally dependent on attribute(s) B if: given a value b for B there is one and only one corresponding value a for A (at a time).



Functional Dependence

Formal concepts that may be used to exhibit “goodness” and “badness” of individual relational schemas, and describe relationships between attributes

- **Examples of Functional Dependency**
  - Name, DateOfBirth, Dept# all depend on NI#
  - Dname & Manager depend on Dept#
  - ProjLocation depends on ProjName.

Functional Dependence

- An attribute, X, of a relation is functionally dependent on attributes A, B, . . ., N if the same values of A , . . . , N are always associated with the same value of X

  {A,…,N}→ X

  - A, .., N is called the **determinant** of the functional dependency
  - This is a property of the **meaning** of the data, not a property that emerges from the values of the data.
  - i.e. if you happened to have no two employees with the same name, you may indeed be able to infer age from name but this would not constitute genuine dependence.

    NI# → {Name, DateOfBirth}

    {NI#,Pnum} → Hours

Full Functional Dependency

- X **fully depends** on A , . . , N if it is not dependent on any subset of A , . . . , N; otherwise we talk of **partial dependency**
  - e.g. age is dependent on NI# and Name but only fully dependent on NI#.

Normalization

- Process
  - taking a set of relations and decomposing them into more relations satisfying some criteria.
- Decomposition
  - essentially a series of projections so that the original data can be reconstituted using joins.
- **Normal form**
  - form of the relations which satisfy the criteria
  - number of these of increasing stringency.

Normal Forms

- A set of conditions on table structure that improves maintenance. Normalization removes processing anomalies:
  - Update
  - Inconsistent Data
  - Addition
  - Deletion

First Normal Form

- A relation is in **first normal form** (1NF) if all values are **atomic**, i.e. single values - small strings and numbers.

- Characteristics
  - All key attributes (columns or fields) are defined
  - All attributes are dependent on the primary key (unique identifier)
  - May have composite keys

First Normal Form – how?

- Identify and remove repeating groups (multi-valued attributes)

- Example: {} ==> repeating group

(A, B, C, {D})

1NF (A, B, C)

(A, D)

First Normal Form – How?

- Eliminate repeating groups
- Often this means completing entries of data in the table

P-No,	P-Name,	E-No,	E-Name,	Job-Class,	Job-$-Rate,	Hours
1	Toys 101	John News		Eng.	65	13
	Toys 102	David Talk		Com.	60	16
	Toys 103	Ann Smith		Prog.	55	19
2	Books101	John News		Eng.	65	24
	Books103	Ann Smith		Prog.	55	44
2	Books104	Tom Jones		Com.	60	11

First Normal Form – How?

- Eliminate repeating groups
- Often this means completing entries of data in the table

P-No,	P-Name,	E-No,	E-Name,	Job-Class,	Job-$-Rate,	Hours
1	Toys 101	John News		Eng.	65	13
1	Toys 102	David Talk		Com.	60	16
1	Toys 103	Ann Smith		Prog.	55	19
2	Books101	John News		Eng.	65	24
2	Books103	Ann Smith		Prog.	55	44
2	Books104	Tom Jones		Com.	60	11

First Normal Form

DEPARTMENT

Dnumber	Dname	Locations
5	C.S.	{ Paris,London }

Two ways of **normalising** this:

- Have a tuple for each location of each department:

Dnumber	Dname	Locations
5	C.S.	Paris
5	C.S.	London

First Normal Form

DEPARTMENT

Dnumber	Dname	Locations
5	C.S.	{ Paris,London }

Two ways of **normalising** this:

- Have a separate relation for (Dnumber, Locations) pairs:

Dnumber	Dname
5	C.S.

Dnumber	Locations
5	Paris
5	London

The latter is better as it avoids redundancy.

Second Normal Form

- By the definition of the primary key, every other attribute is functionally dependent on it.
- If all the other attributes are **fully** functionally dependent then the relation is in **Second Normal Form** (2NF).
- Clearly, **any relation with a single primary key will be 2NF**.
- What about composite keys?

Second Normal Form

- If there are two primary key attributes, A & B, then each other attribute is either
  - dependent on A alone;
  - dependent on B alone;
  - or dependent on both.

2NF Normal consists of creating a separate relation for each of the three cases.

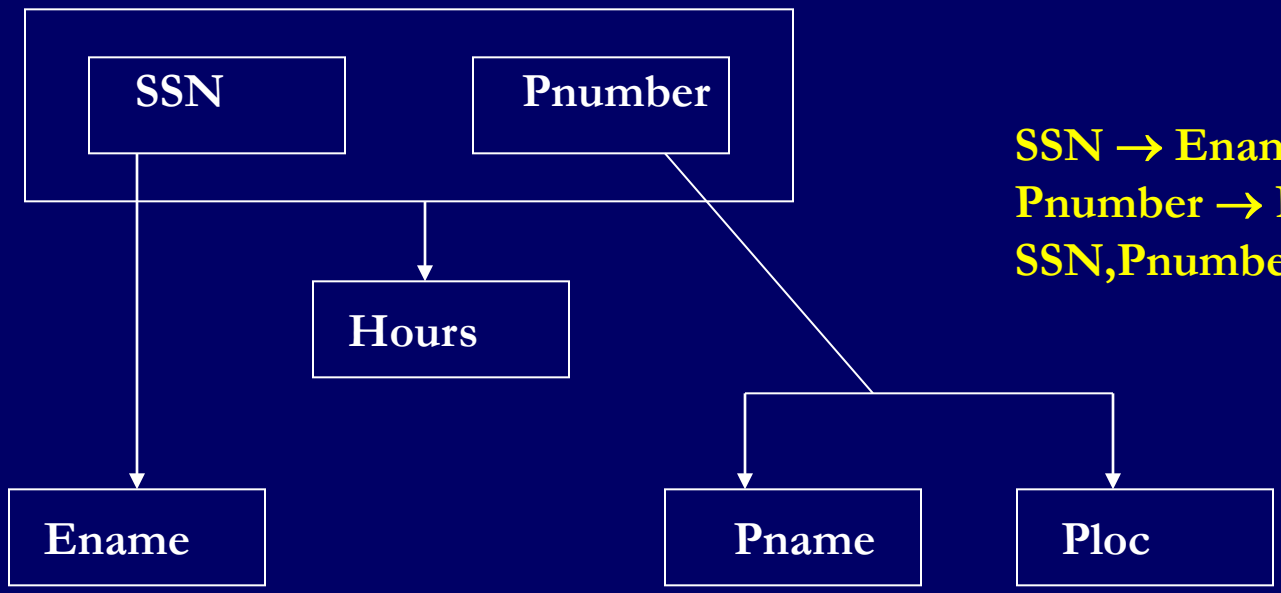
Second Normal Form – how?

- Identify Composite Keys (no Unique Identifier in table)
- Split composite keys (composite unique identifiers) and dependent attributes (columns) into separate tables
- Each part of the composite key becomes the primary key of the new table
- Example:

  (M, N, O, P) where M,N → O & N → P

  2NF: (M, N, O) (N, P)

EMP_PROJ(SSN, Pnumber, Hours, Ename, Pname, Ploc)



SSN → Ename
Pnumber → Pname,Ploc
SSN,Pnumber → Hours

Decomposition into three 2NF relations:

Work(SSN,Pnumber,Hours)
EMP(SSN,Ename)
Project(Pnumber,Pname,Ploc)

Third Normal Form

- **Third Normal Form** eliminates **transitive dependencies** - i.e. those dependencies which hold only because of some intermediary.
  - No transitive dependencies (no non-key field is dependent on another non-key field)

- An attribute is transitively dependent on the primary key if there is some other attribute which it is dependent on and which is, in turn, dependent on the key.

- Example: **(X, Y, Z)** where X → Y & Y → Z

  3NF: **(X, Y)** **(Y, Z)**

Third Normal Form

- **How?**
  - Establish a separate table for any situations where a non-key field is dependent on another non-key field
  - The independent non-key field of the original table is the key field of the new table

Third Normal Form

NI#	Dnumber	Dname
1234	5	C.S.
1235	5	C.S.

Dname is dependent on NI# as required by 2NF, but only because it is dependent on Dnumber which is, in turn, dependent on NI#.

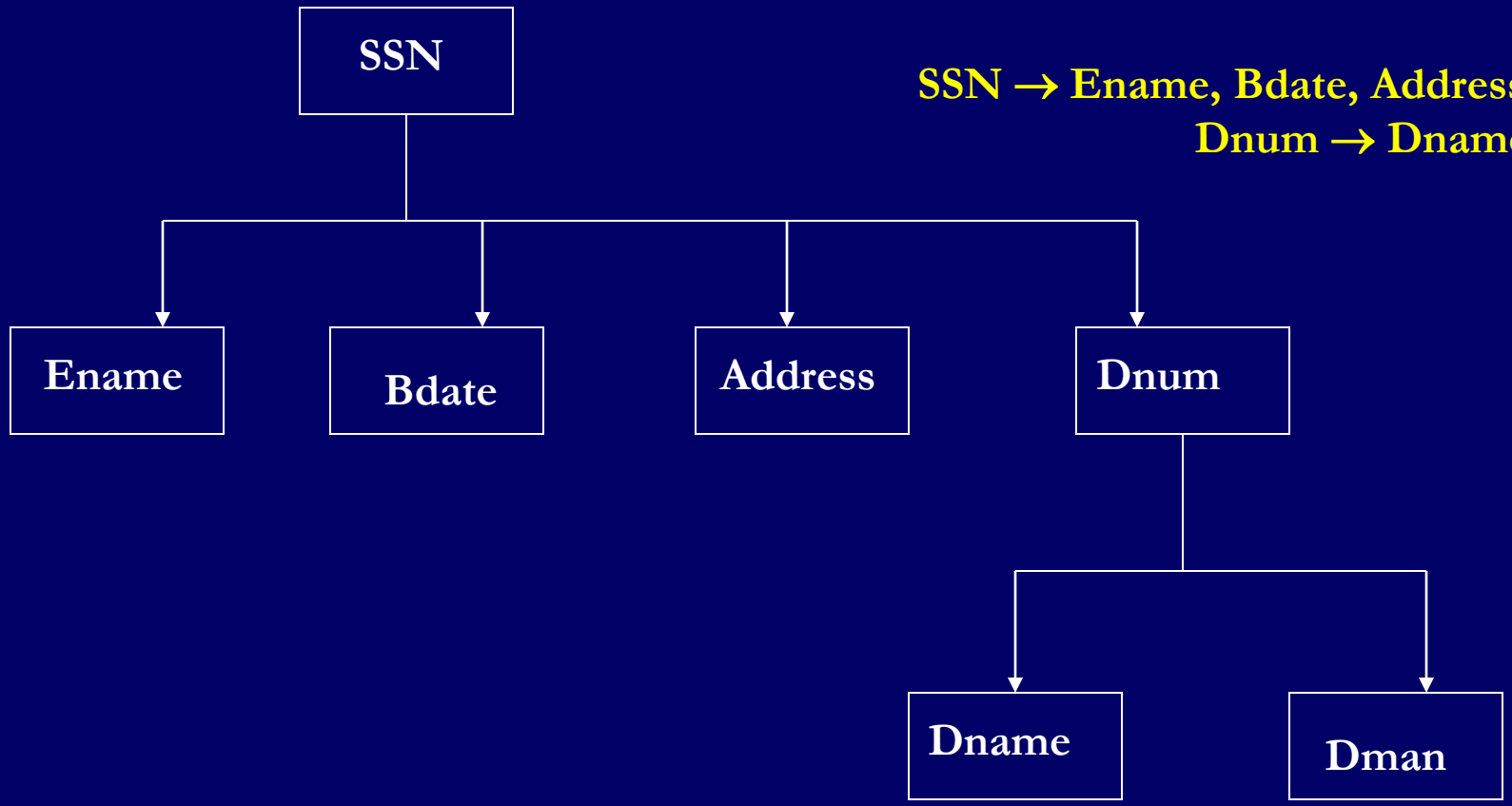
Normalizing this would create:

NI#	Dnumber
1234	5
1235	5

Dnumber	Dname
5	C.S.

- Non-3NF relations are likely to hold redundant information.
- A relation is in 3NF if for any pair of attribute A & B such that A $\rightarrow$ B, there is no attribute such that A $\rightarrow$ X **and** X $\rightarrow$ B.

EMP_DEPT(SSN, Ename, Bdate, Address, Dnum, Dname, Dman)



SSN → Ename, Bdate, Address, Dnum
Dnum → Dname, Dman

Decomposition into two 3NF relations:

EMPLOYEE(SSN,Ename,Bdate,Address,Dnum)
DEPT(Dnum,Dname,Dman)

Boyce-Codd Normal Form (BCNF)

- 2NF + 3NF: no partial dependencies + no transitive dependencies

Normalization Example 1: patient and hospital

Example: Medical Records

- We have:
  - patients with unique NI#, each having a name and a GP;
  - each GP has an id.number, name & address;
  - a hospital appointment connects a patient, a date, a hospital and a consultant;
  - a consultant has a phone number and visits one hospital on any given day;
  - a hospital has an address.

U(**NI# , Appdate**, Apptime, PTname, GP# , GPaddress, GPname, Cname, Cphone, Hosp, HospAddress)

What are the functional dependencies?

What are the Functional Dependencies?

- From a patient's name, we can determine the patient's name and GP:

NI# → {PTname, GP#}

- From a GP's ID we determine the GP's name and address:

GP# → {GPname, GPaddress}

What are the Functional Dependencies?

- From a particular patient on a particular day we can determine the consultant information, the time of the appointment and which was the hospital:

  NI#, AppDate → {Cname, AppTime, Hosp}

- Each consultant has one phone number:

  Cname → Cphone

- Each hospital has only one address:

  Hosp → HospAddress

Normalising the Example

- Starting with the Universal Relation we find all kinds of redundancy.

- So moving to 2NF split off those attributes only dependent on part of the primary key:

  Patient (<u>NI#</u> , PTname, GP# , GPaddress, GPname)

  Appt (<u>NI# , AppDate</u>, AppTime, Cname, Cphone, Hosp, HospAddress)

- Patient is 2NF but not 3NF since NI# $\rightarrow$ GP# $\rightarrow$ {GPaddress, GPname}

Normalising the Example – cont'd

- So split off the GP information

  Patient2 (NI# , PTname, GP#)

  GP (GP# , GPaddress, GPname)

- Similarly, Appt becomes

  App (NI# , AppDate, AppTime, Cname, Hosp)

  Con (Cname, Cphone)

  Hospital (Hosp, HospAddress)

Normalising the Example

Final result:

Patient2 (NI# , PTname, GP#)

GP (GP# , GPaddress, GPname)

App (NI# , AppDate, AppTime, Cname, Hosp)

Con (Cname, Cphone)

Hospital (Hosp, HospAddress)