Functional Dependencies and Normalization

Objectives

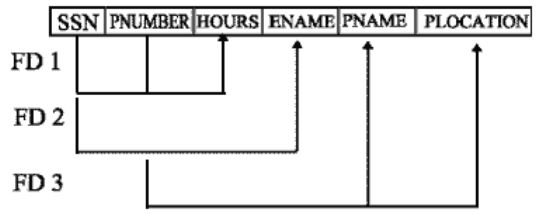
- Understand the concepts of functional dependency and multi-valued dependency
- Understand the concepts of the various normal forms
- Use functional dependencies to express a relational schema in a well-normalized form

Functional Dependency (FD)

- Given a relation, R, and attributes X & Y. The **Functional Dependency** X → Y, between two sets of attributes X and Y that are subsets of R specifies a constraint on the possible tuples that can form a relation instance r of R.
- The constraint is that for any two tuples t_1 and t_2 in r that have the same X values then they must also have the same Y values.

Example

- Consider the relation schema EMP_PROJ, from the semantics of the attributes, we know the following functional dependencies should hold:
  - SSN → ENAME
  - PNUMBER → {PNAME, PLOCATION}
  - {SSN, PNUMBER} → HOURS



Inference Rules for Functional Dependency

- Given a set of functional dependencies, it is possible to infer a set of new dependencies.
- Consider this set of FD's:
  1. STUDNMBR → STUDNAME, BDATE, ADMISSIONDATE, COURSENMBR
  2. COURSENMBR → {CRSNAME, CRSELEADER }
- We can infer from (1) and (2)
  3. STUDNMBR → { CRSNAME, CRSELEADER }
     STUDNMBR → STUDNMBR
     COURSENMBR → CRSNAME

Inferences Rules

- Inferences Rules are a set of rules that tell us how to infer new dependencies from old.
  1. **Reflexive Rule**: If Y is a subset of X then X → Y
  2. **Augmentation Rule**: If X → Y, then {X,Y} → {Y, Z}
  3. **Transitive Rule**: If X → Y and Y → Z, then X → Z
  4. **Decomposition Rule**: If X → {Y, Z} then X → Y and X → Z
  5. **Union Rule**: If X → Y and X → Z, then X → {Y, Z}
  6. **Pseudotransitive Rule**: If X → Y and {W, Y} → Z, then {W, X} → Z

Reflexive Rule

- **If Y is a subset of X then X → Y**
- The reflexive rule tells us that a set of attributes always determines itself or any of its subsets.
- This is obviously true.

Augmentation Rule

- **If X → Y, then {X,Y} → {Y, Z}**
- The augmentation rule says that adding the same set of attributes to both the left and right sides of a dependency will give another valid dependency.

Transitive Rule

- **If X → Y and Y → Z, then X → Z**
- The Transitive Rule tells us that dependencies are transitive.

Decomposition Rule

- **If X → {Y, Z} then X → Y and X → Z**
- The decomposition rule tells us that we can remove attributes from the right hand side of a dependency

Union Rule

- **If X → Y and X → Z, then X → {Y, Z}**
- The union rule allows us to remove attributes from the left hand side of a dependency

Pseudotransitive Rule

- **If X → Y and {W, Y} → Z, then {W, X} → Z**
- the Pseudotransitive Rule tells us that we can combine dependencies, even when transitivity is not formally present.

Armstrong's Inference Rules

- The rules 1-3 are known as Armstrong's Inference Rules which are both sound and complete.
  1. Reflexive Rule: If Y is a subset of X then X → Y
  2. Augmentation Rule: If X → Y, then {X,Y} → {Y, Z}
  3. Transitive Rule: If X → Y and Y → Z, then X → Z

Sound and Complete

- By **Sound** we mean that if we start with a set of functional dependencies, F, and apply those three rules to F, any dependency we obtain will hold in every relation state that satisfies the dependencies in F.
- By **Complete** we mean that if we use the three rules over and over again until we can find no more dependencies we will have the complete set of dependencies that can be inferred from F.

Closure

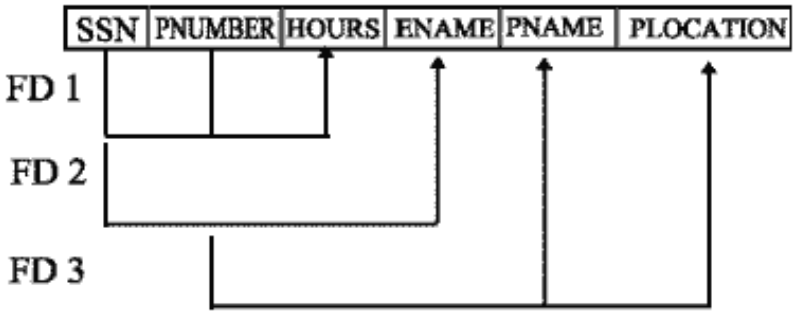
- The **Closure** of a set of dependencies is a set of all functional dependencies that are implied by a given set (F) of FD's.
- We denote this set by F^+.
- F^+ can be found from F by using Armstrong's rules.

Determine Closure of X

- Algorithm to determine closure of X
  - $X^+ := X$
  - repeat
    - $oldX^+ := X^+$
    - for each FD Y → Z in F do
    - if Y is a subset of X^+ then $X^+ := X^+ \cup Z$
  - until ($X^+ = oldX^+$);

Example

- Consider the relationship schema EMP_PROJ



Example (cont')

- From the semantics (meaning) of the attributes, we specify the following set F of functional dependencies that should hold on EMP_PROJ:
  - F = {SSN → ENAME, PNUMBER → {PNAME, PLOCATION}, {SSN, PNUMBER} → HOURS}
- Using the above algorithm, we can calculate the following closure sets with respect to F:
  - $\{SSN\}^+ = \{SSN, ENAME\}$
  - $\{PNUMBER\}^+ = \{PNUMBER, PNAME, PLOCATION\}$
  - $\{SSN, PNUMBER\}^+ = \{SSN, PNUMBER, ENAME, PNAME, PLOCATION, HOURS\}$

Cover of FD

- Let F and E be two sets of functional dependencies. If every FD implied by F is implied by the FD in E, i.e., F^+ is a subset of E, then E is a **Cover** of F.

Equivalent Closures

- If E is a cover of F and F is a cover of E, then F and E are said to be **Equivalent,** i.e., $F^+ = E^+$

Minimal Sets of FD

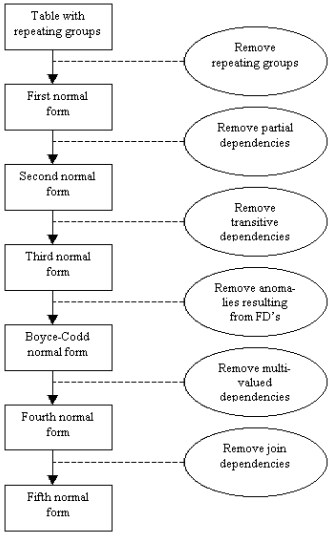
- A set of FDs, F, is **Minimal** (or **Irreducible**) if it satisfies the following:
  - Every dependency in F has a single attribute for its right hand side.
  - The left hand side (the determinant) of every FD in F is minimal in turn. This means that no attribute can be discarded from the left hand side without changing the closure of F
  - No FD in F can be discarded from F without changing the closure of F

Normalization

- Normalization is a process of converting complex data structures into simple, stable data structures.

Steps in Normalization

- Normalization is often accomplished in stages, each of which corresponds to a normal form.
- Normal form is a state of relation that can be determined by applying simple rules regarding dependencies (or relationships between attributes) to that relation.



Steps in Normalization

- Step 1: First normal form (1NF).
  - Any repeating groups have been removed, so that there is a single value at the intersection of each row and column of the table.
- Step 2: Second normal form (2NF).
  - Any partial functional dependencies have been removed.
- Step 3: Third normal form (3NF).
  - Any transitive dependencies have been removed.
- If a relation meets the criteria for 3NF, it also meets criteria for 2NF and 1NF. Most design problems can be avoided if the relations are in 3NF.

Example

- Given a Table in UNF
  - (Order-no, Date, {Part-no, Qty-ordered, Part-description, Quote-price}, Cust-no, Cust-name, Cust-address)
- Since (Part-no, Part-description, Quote-price, etc…) is the repeated group, we need to remove it.

Result of 1NF

UNF	1NF	
Order-no	Order no	Order-no
Date	Date	Part-no
Part-no	Cust-no	Qty-ordered
Qty-ordered	Cust-name	Part-description
Part-description	Cust-address	Quote-price
Quote-price		
Cust-no		
Cust-name		
Cust-address		

Second Normal Form

- A relation is in 2NF if
  - It is in 1NF, and
  - All non-key attributes are fully functionally dependent on the primary key and not on only a portion of the primary key.

Steps to Transform into 2NF

- Identify all functional dependencies in 1NF.
- Make each determinant the primary key of a new relation.
- Place all attributes that depend on a given determinant in the relation with that determinant as non-key attributes.

Example

- All the functional dependencies in this case are:
  - ORDER-NO → DATE, CUST-NO, CUSTNAME, CUST-ADDRESS
  - PART-NO → PART-DESC
- In this case, we say that PART-NO is only partially functional dependent on the key.
  - (ORDER-NO, PART-NO) → QTY-ORDERED, QUOTE-PRICE

Example

- The partial functional dependency in ITEM (ORDER-NO, PART-NO, QTY-ORDERES, QUOTE-PRICE) creates redundancy in that relation, which results in anomalies when the table is updated.

Result for 2NF

1NF		**2NF**	
<u>Order-no</u>	<u>Order-no</u>	<u>Order-no</u>	<u>Order-no</u>
Date	<u>Part-no</u>	Date	<u>Part-no</u>
Cust-no	Qty-ordered	Cust-no	Qty-ordered
Cust-name	Part-description	Cust-name	Quoted-price
Cust-address	Quoted-price	Cust-address	
			<u>Part-no</u>
			Part-description

Note of 2NF

- A relation that is in first normal form will be in second normal form if any one of the following conditions apply:
- The primary key consists of only one attribute (such as the attribute ORDER-NO in ORDER).
- No non-key attributes exist in the relation.
- Every non-key attribute is functionally dependent on the full set of primary key attributes.

Third Normal Form

- A relation is in 3NF if:
  - It is in 2NF, and
  - No transitive dependencies.
- Transitive dependencies are when A → B → C. Thus it can be split into A → B and B → C.

Steps to Transform into 3NF

- Create one relation for each determinant in the transitive dependency.
- Make the determinants the primary keys in their respective relations.
- Include as non-key attributes those attributes that depend on the determinant.

Example

- In the functional dependency:
  - ORDER(ORDER-NO, DATE, CUST-NO, CUST-NAME, CUST-ADDRESS)
- There is a transitive dependency.
- That is, one of the non-key attribute can be used to determine other attributes.
  - CUST-NO → CUST-NAME, CUST ADDRESS

Result of 3NF

- Notice that CUST-NO is the primary key of a new relation and is a foreign key in the ORDER relation.

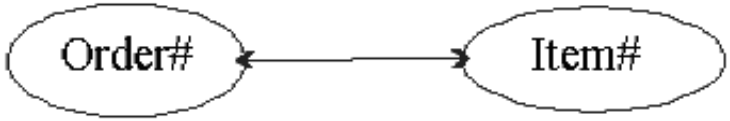
2NF		**3NF**	
Order-no Date Cust-no Cust-name Cust-address	Order-no Part-no Qty-priced Quoted-price Part-no Part-description	Order-no Date Cust-no Cust-no Cust-name Cust-address	Order-no Part-no Qty-ordered Quoted-price Part-no Part-description

Dependency Diagrams

- Suppose we have four different enterprises which allow Orders to be made for certain Items.
- The Orders are identified by an Order# and the Items by an Item#.
- There are four possibilities.

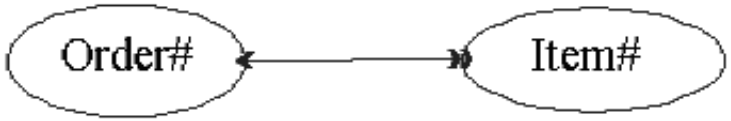
Possibility 1

- Each Order is for one Item only and an Item can only be placed in one Order,
  - i.e. Order# : Item# is 1:1



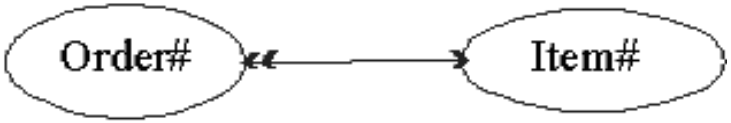
Possibility 2

- Each Order can be for many items, however an Item can only be on one Order,
  - i.e. Order# : Item# is 1:M



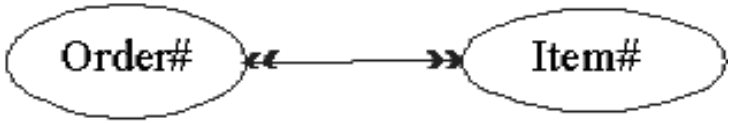
Possibility 3

- Each Order is for one Item only, however an Item can be placed on many Orders,
  - i.e. Order# : Item# is M:1



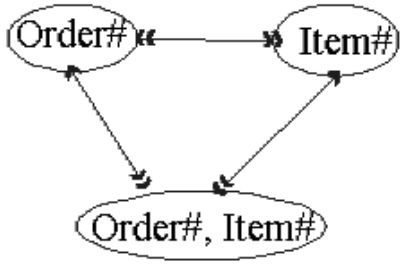
Possibility 4

- Each Order can be for many Items and an Item can be placed on many Orders,
  - i.e. Order# : Item# is M:M



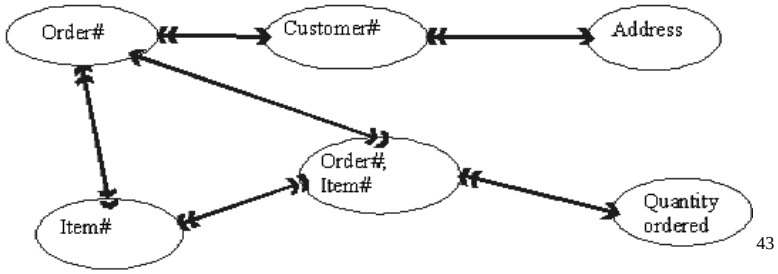
Dependency Diagrams

- In this last case, we form a new composite attribute combining both of the original attributes.
- This will then be in a many to one relationship with each of the original attributes:



Example

- The following diagram represents several enterprise rules, abbreviated to the FDs:
  - Order# determines Customer#
  - Customer# determines Address
  - {Order#, Item#} determines Quantity ordered



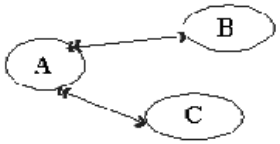
Full Functional Dependency

- B is fully functionally dependent on A if A determines B and no subset of A determines B.
- Now once we have drawn our dependency diagrams it is very easy indeed to spot the determinants.

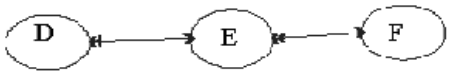
Rules of Determinants

- Determinants must not include superfluous attributes
- Determinants can be composite attributes
- Determinants are transitive

Spotting Determinants from Dependency Diagrams

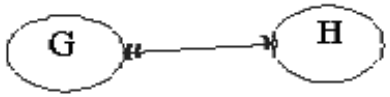


Determinant = A
B and C are dependent on A

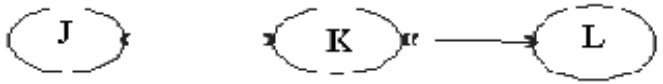


Determinants = D and E
E is dependent on D
F is dependent on E

Spotting Determinants from Dependency Diagrams

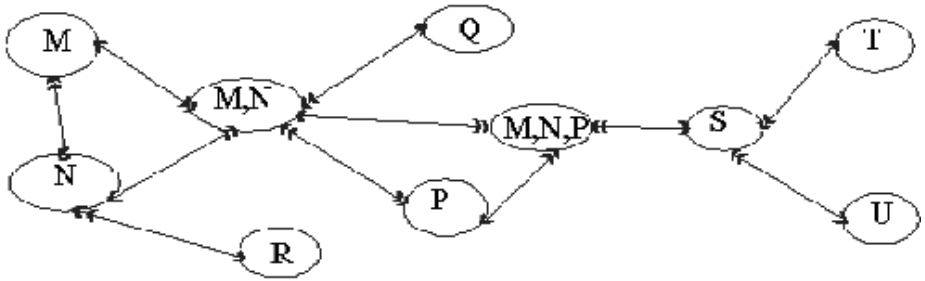


No functional dependencies



Determinants = J and K
K is dependent on J
J and L are dependent on K

Spotting Determinants from Dependency Diagrams



Determinants = {M, N, P}, {M, N}, N and S
Q s dependent on {M, N}
S is dependent on {M, N, P}
T and U are dependent on S

Identifier

- Every relation has some combination of attributes that will unambiguously identify each tuple.
- This is the **Identifier**. It cannot be null, must not contain superfluous attributes and can never take duplicate values. We show an identifier by underlining it.

Example

- ORDER (Order#, Item#, Quantity ordered, Customer#, Address) has identifier { Order#, Item# }

ORDER

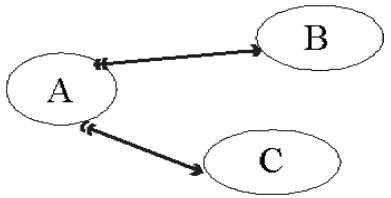
Order#	*Item#*	*Quantity ordered*	*Customer #*	*Address*
O200	100	500	C1	Singapore
O200	102	500	C1	Singapore
O201	700	250	C2	Hong Kong
O202	102	520	C1	Singapore
O202	700	250	C1	Singapore
O202	444	400	C1	Singapore

Deduce Identifier from Dependency Diagram

- From a dependency diagram we can deduce what the identifier would be if we were to put all the data elements into one relation.

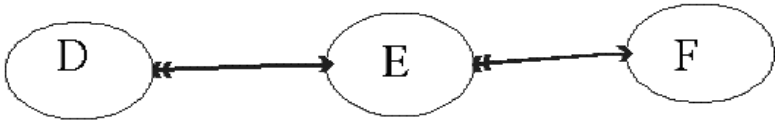
Example

- Relation is {A, B, C}



- The diagram tells us that given a value of A, we can determine one associated value for B and one associated value for C. So a value of A is sufficient to identify a complete tuple in the relation. As A cannot contain any superfluous attributes, A must be an identifier.

Example

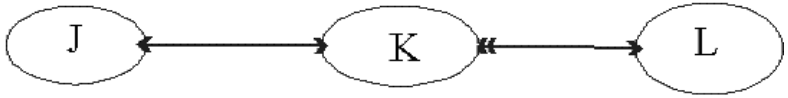


- Relation is {D, E, F}

Example

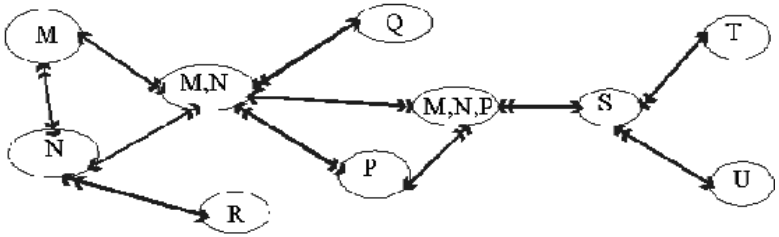
- Relation is {G, H}

Example



- Relation is {J, K, L} or {J, K, L}

Example

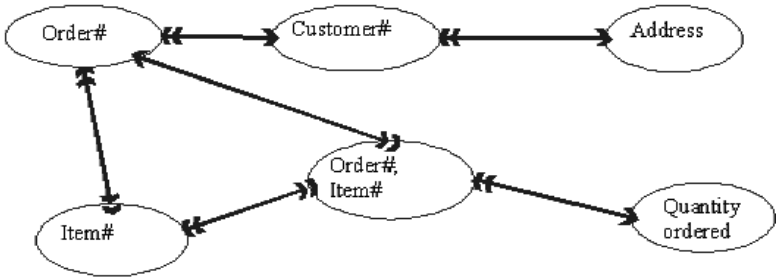


- Relation is {M, N, P, Q, R, S, T, U}

Boyce/Codd rule

- We can find redundancy in a relation by using the *Boyce/Codd rule*:
  - **"Every determinant must be a candidate identifier."**
- A relation that obeys this rule is in **Boyce/Codd Normal Form** (BCNF). We say such a relation is **well-normalized**.
- To transform an un-normalized relation into well-normalized relations, create new relations so that each non-identifying determinant (i.e. a determinant which is not a candidate identifier) in the old relation becomes a candidate identifier in a new relation.

Example



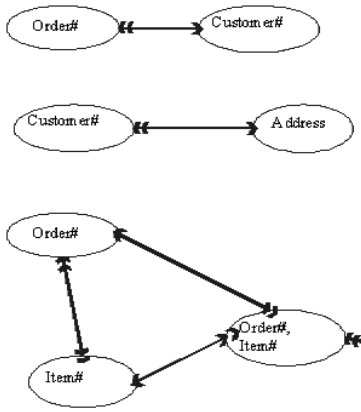
- The determinants are: Order#, Customer# and {Order#, Item#}.

Example (Cont')

- With a relation of {Order#, Item#, Customer#, Address, Quantity ordered} the identifier would be {Order#, Item#}.
- So Customer# and Order# are non-identifying determinants.

Example (Cont')

- Hence we create two new relations and our well-normalized set of relations will be:



Example (Cont')

- And the associated sample data will look like:

ORDER

Order #	Customer #
O200	C1
O201	C2
O202	C1

CUSTOMER

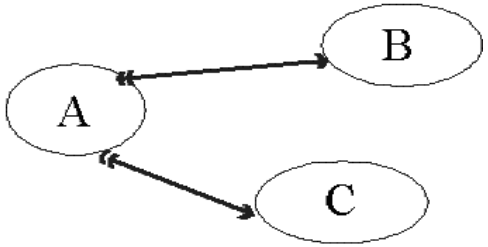
Customer #	*Address*
C1	Singapore
C2	Hong Kong

ORDER LINE

Order #	*Item #*	*Quantity ordered*
O200	100	500
O200	102	500
O201	700	250
O202	102	520
O202	700	250
O202	444	400

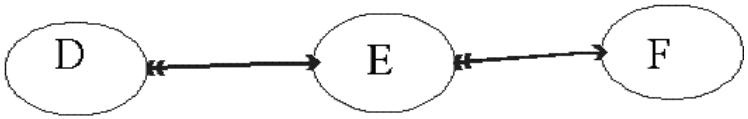
Example

- If we now look at the set of dependency diagrams that we have been using before and apply the Boyce/Codd rule to them we will get the following well-normalized relations:
  - Relation is {A, B, C}
  - Determinant A becomes the identifier



Example

- Relations are {D, E} and {E, F}
- Determinants D and E become relation identifiers

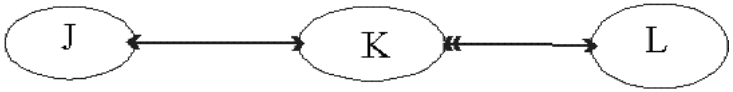


Example

- Relation is {G, H}
- No determinants

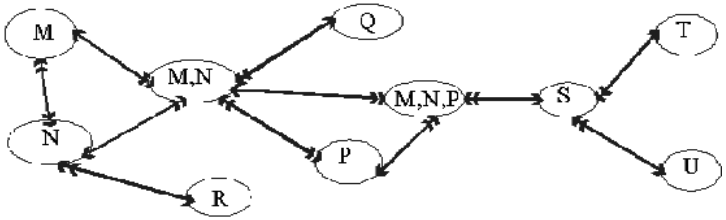
Example

- Relations are {J, K} or {J, K} and {K, L}
- Determinants J and K become identifiers



Example

- Relations are {M, N, P, S} , {M, N, Q}, {N, R} and {S, T, U}
- Determinants {M,N,P}, {M, N}, N and S become identifiers



Summary

- Following the Boyce/Codd rule ensures that all relations are well normalized.
- All transitive dependencies are eliminated by applying this rule.
- Any relation in BCNF is also in 3NF.
- However such relations may still contain redundancy due to multi-valued dependencies.

What is a dependency?

- A dependency is a statement that only a subset of all the possible relation instances are legal;
  - i.e. only certain combinations of values of attributes reflect a possible state of the real world.
- It follows that if not all relation instances are possible then there will be some sort of redundancy in legal relations;
  - i.e. given a legal relation R, which satisfies certain dependencies and if we have certain information about the current value of R, we should be able to deduce other things about the current value of R.
- Informally we might say "if you see a certain pattern you must also see this."

Functional Dependency

- **A determines B** if, given a particular value of A, this specifies a unique value for B.
- If we have a relation R(A,B,C,D) with attribute values, then we can conclude that b1 = b2.

A	B	C	D
a	b_1	c_1	d_1
a	b_2	c_2	d_2

Multivalued Dependency

- **A multi determines B** if, given a particular value of A, the set of values of B determined by this value of A is completely determined by this value of A alone and does not depend on the values of the remaining attributes of the relation schema.
- If we have a relation R (A,B,C,D) with attribute values, then we can conclude that the tuples (a, b1, c2, d2) and (a, b2, c1, d1) must also belong to R.

A	B	C	D
a	b_1	c_1	d_1
a	b_2	c_2	d_2

Example 1

- Suppose that an employee may work on several projects and may have several dependents.
- The projects and dependents will not, of course, be related to each other.
- The following relation, which is in BCNF, models this situation:

EMP

Ename	*Pname*	*Dname*
Gayfer	Telecoms	Harold
Gayfer	Accounts	Mary
Gayfer	Telecoms	Mary
Gayfer	Accounts	Harold

Example 1 (cont')

- Ename multidetermines Pname and Dname. Although the relation is in BCNF, there is obvious redundancy here!
  - For example if Gayfer begins work on another project, Overseas-mergers, say, we would have to add two new tuples to the relation.

Example 1 (cont')

- If we split the relation into two, we eliminate the redundancy.
- To add the information about Gayfer working on Overseas-mergers now only takes one new line. These two relations are now in 4NF.
- Whenever two independent 1:N relationships A:B and A:C are mixed in the same relation by representing all possible combinations, an MVD may arise.

EMP-PROJECTS	Ename	Pname
	Gayfer	Telecoms
	Gayfer	Accounts

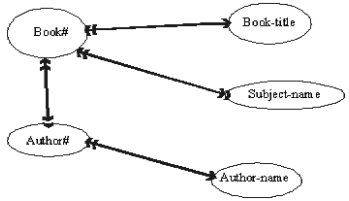
EMP-DEPTS	Ename	Dname
	Gayfer	Harold
	Gayfer	Mary

Example 2

- Suppose we want to model the authors and subject classifications of books. Each book has a unique book#, each author a unique author# and each subject classification a unique subject-name. Book# does not distinguish an individual copy of a book, rather it distinguishes an individual work. A book may be written by several authors and be classified under several subject-names. An author may write several books. A subject-name may be applied to several books. In addition to book#, author# and subject-name, other attributes of interest are author-name and book-title.

Example 2 (cont')

- The dependency diagram is shown below. Also shown is an occurrence of the Author-Book-Subject relation.



Book#	Author#	Subject-name
B15	A2	Economics
B15	A2	Computing
B15	A5	Economics
B15	A5	Computing
B18	A2	Computing

Example 2 (cont')

- If it is true that every author of a given book is always associated with all the subject-names under which the book is classified, then we have a MVD. Book# multi-determines author# and subject-name. If the row (B15, A2, Economics) were omitted from the relation we could deduce its existence from the rows (B15, A2, Computing) and (B15, A5, Economics). We can split the relation into the two following ones:

Book#	Author#
B15	A2
B15	A5
A18	A2

Book#	Subject-name
B15	Economics
B15	Computing
B18	Computing

Example 2 (cont')

- These are now in 4NF.
- However if subject-name refers to a subject area within a book for which an individual author was responsible we no longer have an MVD and the original three attribute relation would be in 4NF.
- The difference now is that Book# and Author# together determine subject-name, so the dependency diagram will be different, (try drawing it) and we no longer are mixing two independent 1:N relationships.

Join Dependency

- Whenever we decompose a relation schema R into several relations R_1, R_2,...and then form the join of those relations R_1, R_2,.. we expect that no spurious tuples will be generated.
- This is a property of relation schemas, so the condition of no spurious tuples should hold on every legal instance,
  - i.e. every relation that satisfies the dependencies specified on the schema.
- This is called the **Lossless Join Property** or **Nonadditive Join Property**.

Join Dependency

- If we have a MVD we can decompose our relation schema into two schemas so that they satisfy this lossless join property. However there are cases when a relation schema cannot be decomposed into only two schemas that satisfy this property but can be decomposed into more than two schema which do.
- In such cases we have what is known as a **Join Dependency** (JD).

Join Dependency

- A **Join Dependency** specified on a relation schema R, specifies a constraint on the instances of R.
- The constraint states that *every legal instance* of R should have a lossless join decomposition into R_1, R_2,....,R_n, that is, if we only use legal instances in R then project to R_1,...,R_n and then join those projected relation instances, no illegal instances can result.
- Once we have decomposed R into those relations R1,...,Rn then our schema is in **Fifth** (or **Project-Join**) **Normal Form**.

Example

- Consider the relation schema SUPPLY (Sname, Partname, Projname) with the constraint
- Whenever a supplier S supplies part P and a project J uses part P and the supplier S supplies at least one part to project J, then supplier S will also be supplying part P to project J.
- This constraint will produce a Join Dependency.

Sname	*Partname*	*Projname*
Ocampo	Cement	Office Block
Ocampo	Timber	House
Lee	Cement	House
Singh	Timber	Garage
Lee	Steel	Office Block
Lee	Cement	Office Block
Ocampo	Cement	House

Example (cont')

- We can decompose the Supply relation into three relations as follows:

R1

Sname	*Partname*
Ocampo	Cement
Ocampo	Timber
Lee	Cement
Singh	Timber
Lee	Steel

R2

Sname	*Projname*
Ocampo	Office Block
Ocampo	House
Lee	House
Singh	Garage
Lee	Office Block

R3

Partname	*Projname*
Cement	Office Block
Timber	House
Cement	House
Timber	Garage
Steel	Office Block

Example (cont')

- Taking the natural join of any two of the relations produces illegal tuples, for instance R1 $\triangleright\triangleleft$ R2 would give (Ocampo, Timber, Office Block).
- This tuple is not in the relation, it is not a legal instance of R. However taking the natural join of all three relations does not produce these illegal tuples.
- So the constraint above does specify a join dependency on the three projections.

Example (cont')

- Note that discovering Join Dependencies in practical databases with hundreds of attributes is difficult hence current practice of database design pays scant attention to them.
- However if the model is drawn carefully and relations are constructed so that every determinant becomes a candidate identifier and every multi-valued relationship is put in its own relation, then this will ensure that the resulting relations are fully-normalized.