Data Modeling and Database Design

Chapter 7:
Functional Dependencies

Issue of the Moment

- The “goodness” of design of the data model
- What do we know about the “quality” of the logical schema we have?
- How do we vouch for the goodness of the initial conceptual data model ?
- How do we vouch for the quality of the process of transforming the conceptual model to its logical counterpart ?
- How do we make sure that the database design on hand at this point, if implemented, will work without causing any problems ?
- Grouping of attributes (e.g., entities) has so far been an intuitive process and requires rigorous validation to ensure design quality.

The Issue of Quality of Design: An Illustration

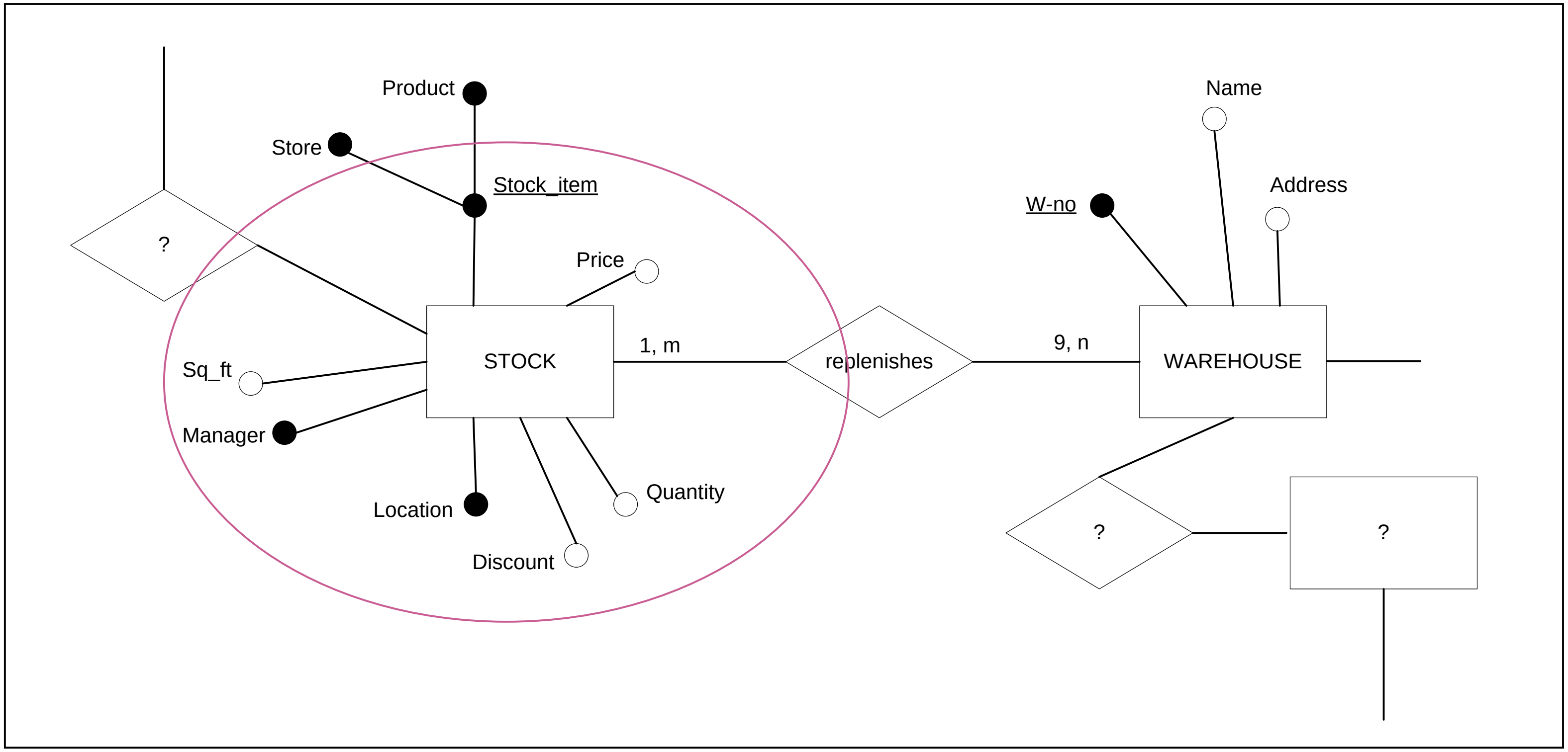


Figure 7.1a An excerpt from an ER diagram

STOCK (Store, Product, Price, Quantity, Location, Discount, Sq_ft, Manager)

Figure 7.1b Relation schema for the entity type, STOCK

A Relational Instance of “STOCK”

STOCK

Store	Product	Price	Quantity	Location	Discount	Sq_ft	Manager
15	Refrigerator	1850	120	Houston	5%	2300	Metzger
15	Dishwasher	600	150	Houston	5%	2300	Metzger
13	Dishwasher	600	180	Tulsa	10%	1700	Metzger
14	Refrigerator	1850	150	Tulsa	5%	1900	Schott
14	Television	1400	280	Tulsa	10%	1900	Schott
14	Humidifier	55	30	Tulsa		1900	Schott
17	Television	1400	10	Memphis		2300	Creech
17	Vacuum Cleaner	300	150	Memphis	5%	2300	Creech
17	Dishwasher	600	150	Memphis	5%	2300	Creech
11	Computer		180	Houston	10%	2300	Creech
11	Refrigerator	1850	120	Houston	5%	2300	Creech
11	Lawn Mower	300		Houston		2300	Creech

Figure 7.1c An instance of the relation schema, STOCK

Is there a data redundancy for the attribute:

Price? Location? Quantity? Discount?

What is Data Redundancy?

- Repeated appearance of same data value for an attribute <u>**does not**</u> automatically mean data redundancy.
- Superfluous repetition that does not add new meaning constitutes data redundancy.
- Error in attribute allocation leads to data redundancy.
- *Data redundancy leads to modification anomalies.*

A Relational Instance of "STOCK"

STOCK

Store	Product	Price	Quantity	Location	Discount	Sq_ft	Manager
15	Refrigerator	1850	120	Houston	5%	2300	Metzger
15	Dishwasher	600	150	Houston	5%	2300	Metzger
13	Dishwasher	600	180	Tulsa	10%	1700	Metzger
14	Refrigerator	1850	150	Tulsa	5%	1900	Schott
14	Television	1400	280	Tulsa	10%	1900	Schott
14	Humidifier	55	30	Tulsa		1900	Schott
17	Television	1400	10	Memphis		2300	Creech
17	Vacuum Cleaner	300	150	Memphis	5%	2300	Creech
17	Dishwasher	600	150	Memphis	5%	2300	Creech
11	Computer		180	Houston	10%	2300	Creech
11	Refrigerator	1850	120	Houston	5%	2300	Creech
11	Lawn Mower	300		Houston		2300	Creech

Figure 7.1c An instance of the relation schema, STOCK

Is there a data redundancy for the attribute:

Price? Yes Location? Yes Quantity? No Discount? Yes

Modification Anomalies

Data redundancy leads to modification anomalies.

- Insertion Anomaly
→ Suppose we want to add “Washing Machine” to our stock with a price.

- Deletion Anomaly
→ Suppose store 17 is closed.

- Update Anomaly
→ Suppose we want to change the location of store 11 from Houston to Cincinnati.

Insertion Anomaly

Suppose we want to add “Washing Machine” to our stock with a price of $600

STOCK

Store	Product	Price	Quantity	Location	Discount	Sq_ft	Manager
15	Refrigerator	1850	120	Houston	5%	2300	Metzger
15	Dishwasher	600	150	Houston	5%	2300	Metzger
13	Dishwasher	600	180	Tulsa	10%	1700	Metzger
14	Refrigerator	1850	150	Tulsa	5%	1900	Schott
14	Television	1400	280	Tulsa	10%	1900	Schott
14	Humidifier	55	30	Tulsa		1900	Schott
17	Television	1400	10	Memphis		2300	Creech
17	Vacuum Cleaner	300	150	Memphis	5%	2300	Creech
17	Dishwasher	600	150	Memphis	5%	2300	Creech
11	Computer		180	Houston	10%	2300	Creech
11	Refrigerator	1850	120	Houston	5%	2300	Creech
11	Lawn Mower	300		Houston		2300	Creech
	Washing Machine	600					

Figure 7.1c An instance of the relation schema, STOCK

The insertion shown is illegal since Store, a proper subset of the primary key of STOCK cannot have a null value. In other words, without a Store for the Washing Machine, it is not possible to add this information – an insertion anomaly.

Deletion Anomaly

Suppose store 17 is closed

STOCK

Store	Product	Price	Quantity	Location	Discount	Sq_ft	Manager
15	Refrigerator	1850	120	Houston	5%	2300	Metzger
15	Dishwasher	600	150	Houston	5%	2300	Metzger
13	Dishwasher	600	180	Tulsa	10%	1700	Metzger
14	Refrigerator	1850	150	Tulsa	5%	1900	Schott
14	Television	1400	280	Tulsa	10%	1900	Schott
14	Humidifier	55	30	Tulsa		1900	Schott
~~17~~	~~Television~~	~~1400~~	~~10~~	~~Memphis~~		~~2300~~	~~Creech~~
~~17~~	~~Vacuum Cleaner~~	~~300~~	~~150~~	~~Memphis~~	~~5%~~	~~2300~~	~~Creech~~
~~17~~	~~Dishwasher~~	~~600~~	~~150~~	~~Memphis~~	~~5%~~	~~2300~~	~~Creech~~
11	Computer		180	Houston	10%	2300	Creech
11	Refrigerator	1850	120	Houston	5%	2300	Creech
11	Lawn Mower	300		Houston		2300	Creech

Figure 7.1c An instance of the relation schema, STOCK

Not only does the action required entails deletion of multiple rows, but also is there an inadvertent loss of information in this case that vacuum cleaner is priced at $300 – a deletion anomaly.

Update Anomaly

Suppose store 11 is moved from Houston to Cincinnati

STOCK

Store	Product	Price	Quantity	Location	Discount	Sq_ft	Manager
15	Refrigerator	1850	120	Houston	5%	2300	Metzger
15	Dishwasher	600	150	Houston	5%	2300	Metzger
13	Dishwasher	600	180	Tulsa	10%	1700	Metzger
14	Refrigerator	1850	150	Tulsa	5%	1900	Schott
14	Television	1400	280	Tulsa	10%	1900	Schott
14	Humidifier	55	30	Tulsa		1900	Schott
17	Television	1400	10	Memphis		2300	Creech
17	Vacuum Cleaner	300	150	Memphis	5%	2300	Creech
17	Dishwasher	600	150	Memphis	5%	2300	Creech
11	Computer		180	Cincinnati	10%	2300	Creech
11	Refrigerator	1850	120	Cincinnati	5%	2300	Creech
11	Lawn Mower	300		Houston		2300	Creech

Figure 7.1c An instance of the relation schema, STOCK

In addition to the necessity to update multiple rows, inadvertent failure to update all the relevant rows changes the semantics of the scenario – i.e., store 11 is located in both Houston and Cincinnati – an update anomaly.

Modification Anomalies (continued)

STORE

Store	Location	Sq_ft	Manager
15	Houston	2300	Metzger
13	Tulsa		Metzger
14	Dallas	1900	Schott
17	Memphis	2300	Creech
11	Houston	1900	Parrish

PRODUCT

Product	Price
Refrigerator	1850
Dishwasher	600
Television	1400
Humidifier	55
Vacuum Cleaner	300
Computer	
Lawn Mower	300
Washing Machine	750

Decomposition of STOCK to eliminate modification anomalies caused by redundant data

INVENTORY

Store	Product	Quantity	Discount
15	Refrigerator	120	5%
15	Dishwasher	150	5%
13	Dishwasher	180	10%
14	Refrigerator	150	5%
14	Television	280	10%
14	Humidifier	30	
17	Television	10	
17	Vacuum Cleaner	150	5%
17	Dishwasher	150	5%
11	Computer	120	5%
11	Refrigerator	180	10%
11	Lawn Mower		

No data redundancy/modification anomalies in STORE and PRODUCT

Data redundancy/modification anomalies persist in INVENTORY

Figure 7.2 A decomposition of the STOCK instance in Figure 7.1c

Modification Anomalies (continued)

STORE

Store	Location	Sq_ft	Manager
15	Houston	2300	Metzger
13	Tulsa	1700	Metzger
14	Tulsa	1900	Schott
17	Memphis	2300	Creech
11	Houston	2300	Creecg

PRODUCT

Product	Price
Refrigerator	1850
Dishwasher	600
Television	1400
Humidifier	55
Vacuum Cleaner	300
Computer	
Lawn Mower	300
Washing Machine	750

Decomposition of INVENTORY to eliminate modification anomalies caused by redundant data

INVENTORY

Store	Product	Quantity
15	Refrigerator	120
15	Dishwasher	150
13	Dishwasher	180
14	Refrigerator	150
14	Television	280
14	Humidifier	30
17	Television	10
17	Vacuum Cleaner	150
17	Dishwasher	150
11	Computer	180
11	Refrigerator	120
11	Lawn Mower	

DISC_STRUCTURE

Quantity	Discount
120	5%
150	5%
180	10%
280	10%
30	
10	

Figure 7.3 A redundancy-free decomposition of the STOCK instance in Figure 7.1c

The design that is free from data redundancies/modification anomalies is said to be "normalized."

Modification Anomalies (continued)

STORE

Store	Location	Sq_ft	Manager
15	Houston	2300	Metzger
13	Tulsa	1700	Metzger
14	Tulsa	1900	Schott
17	Memphis	2300	Creech
11	Houston	2300	Creecg

PRODUCT

Product	Price
Refrigerator	1850
Dishwasher	600
Television	1400
Humidifier	55
Vacuum Cleaner	300
Computer	
Lawn Mower	300
Washing Machine	750

INVENTORY

Store	Product	Quantity
15	Refrigerator	120
15	Dishwasher	150
13	Dishwasher	180
14	Refrigerator	150
14	Television	280
14	Humidifier	30
17	Television	10
17	Vacuum Cleaner	150
17	Dishwasher	150
11	Computer	180
11	Refrigerator	120
11	Lawn Mower	

DISC_STRUCTURE

Quantity	Discount
120	5%
150	5%
180	10%
280	10%
30	
10	

Figure 7.3 A redundancy-free decomposition of the STOCK instance in Figure 7.1c

What is the relational schema that will yield this set of tables which being free of data redundancies and modification anomalies, is said to be “normalized”?

Relational Schema Reverse-Engineered From the Set of Tables

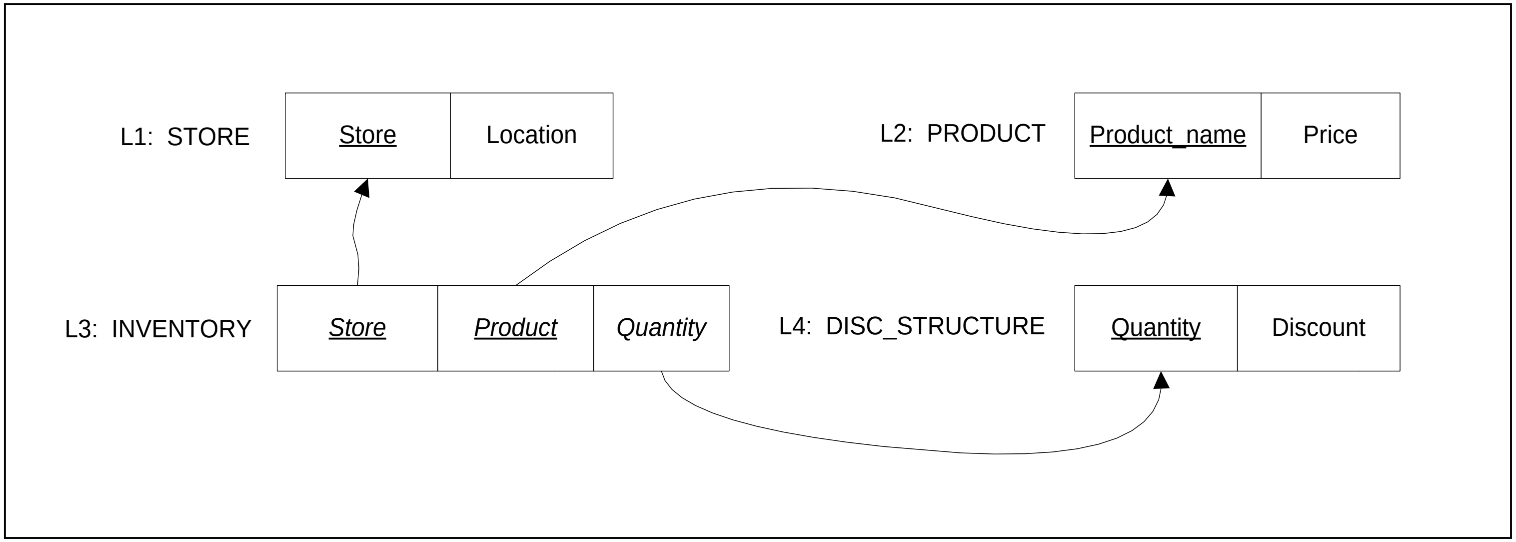


Figure 7.4a A reverse-engineered logical schema for the for the set of tables in Figure 7.3

Design-specific ER Diagram Reverse-Engineered from the Relational Schema

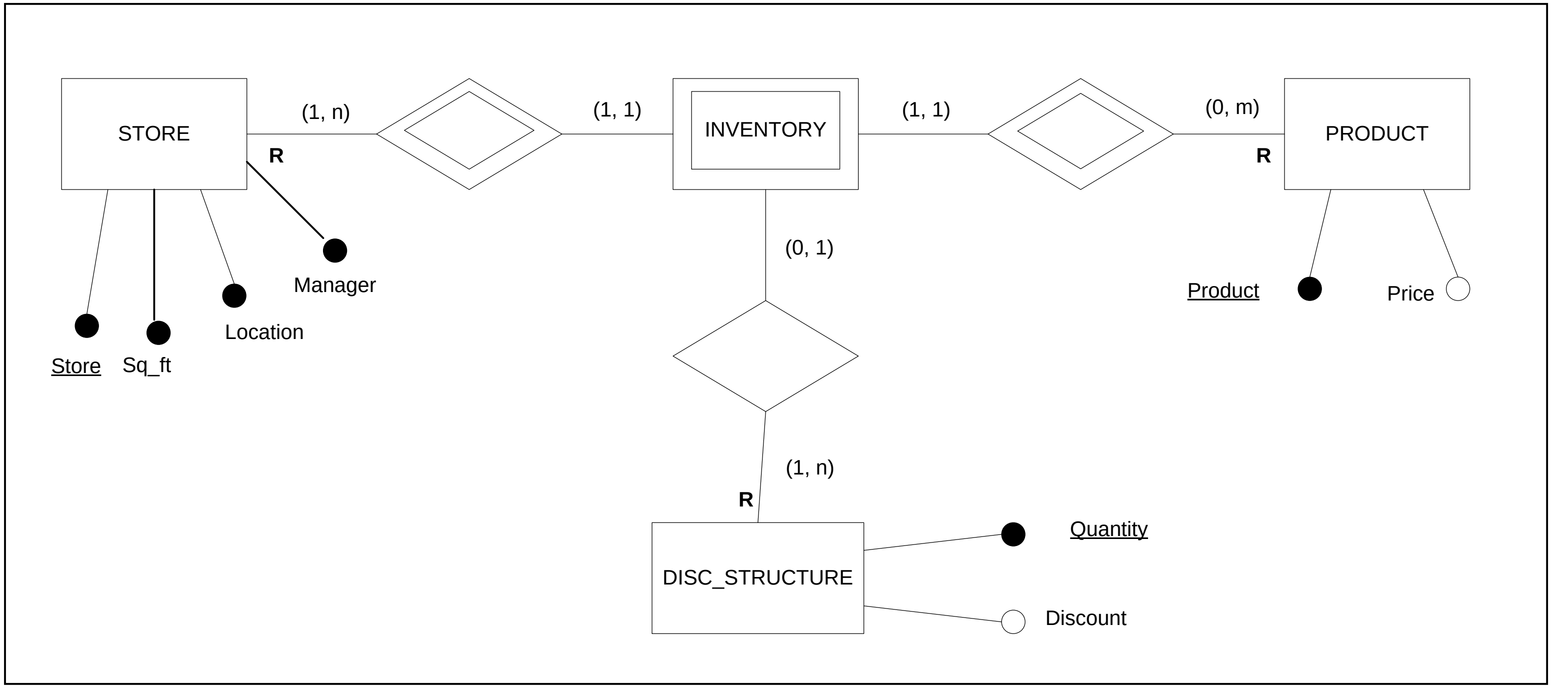


Figure 7.4b Design-specific ER diagram reversed-engineered from the logical schema in Figure 7.4a

Presentation Layer ER Diagram Reverse-Engineered from the Design-specific ER Diagram

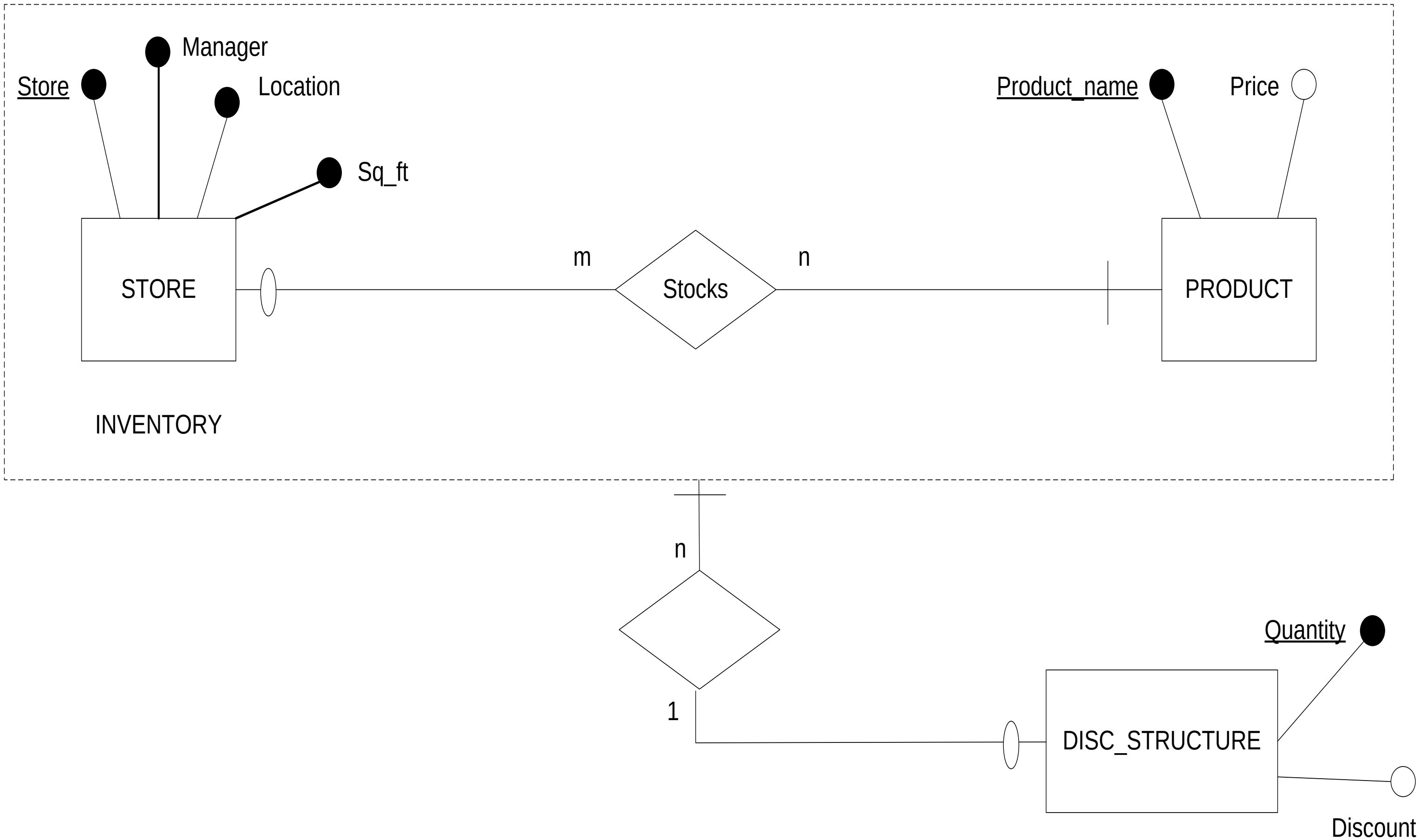


Figure 7.4c Presentation layer ER diagram reversed-engineered from Figure 7.4b

As it was . . . with modification anomalies

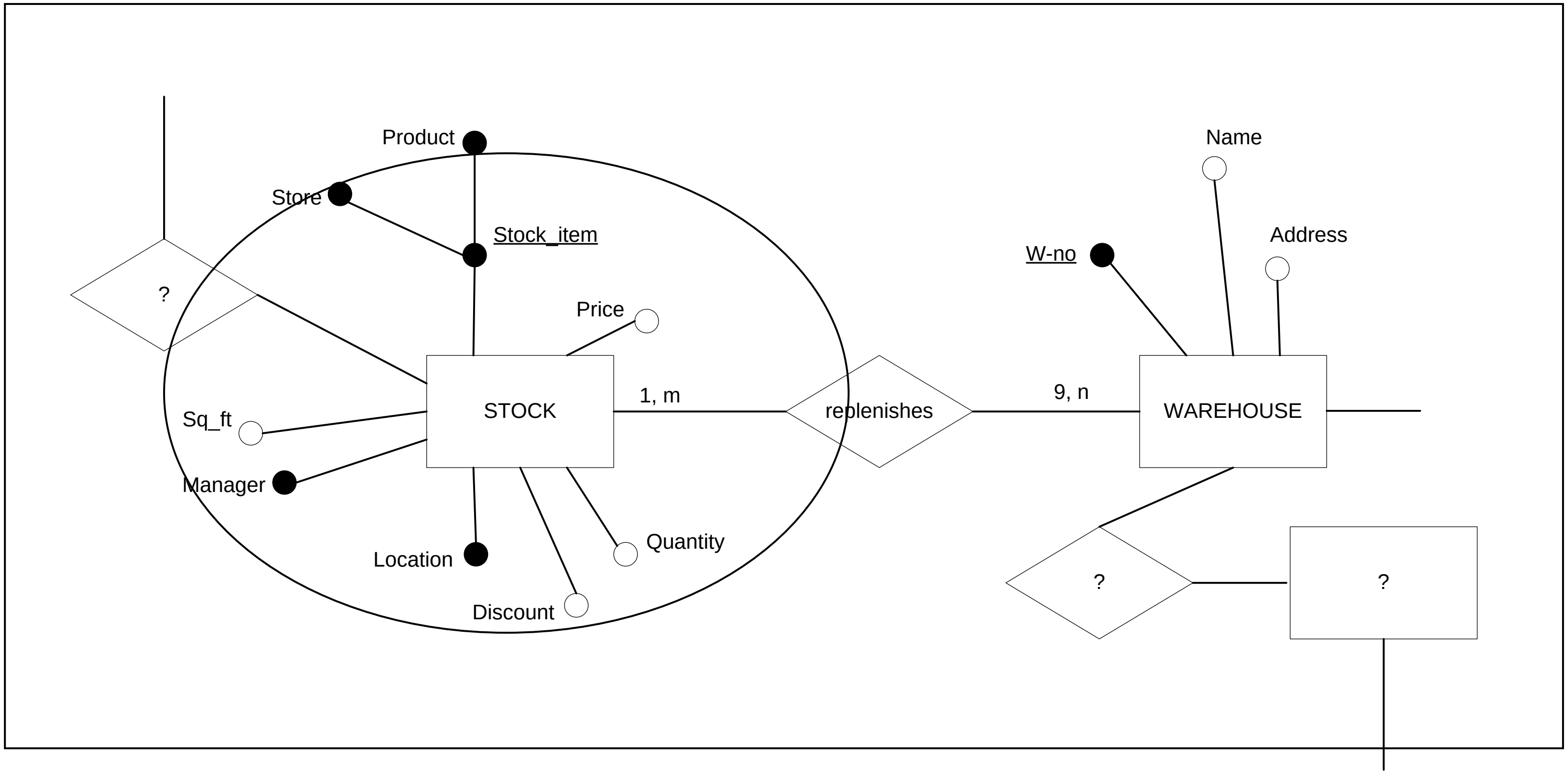


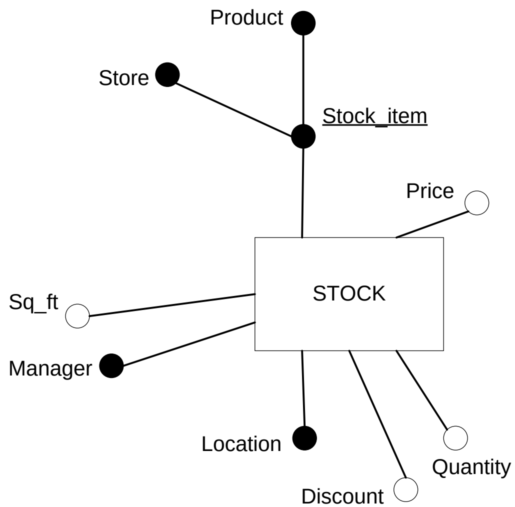
Figure 7.1a An excerpt from an ER diagram

STOCK (Store, Product, Price, Quantity, Location, Discount, Sq_ft, Manager)

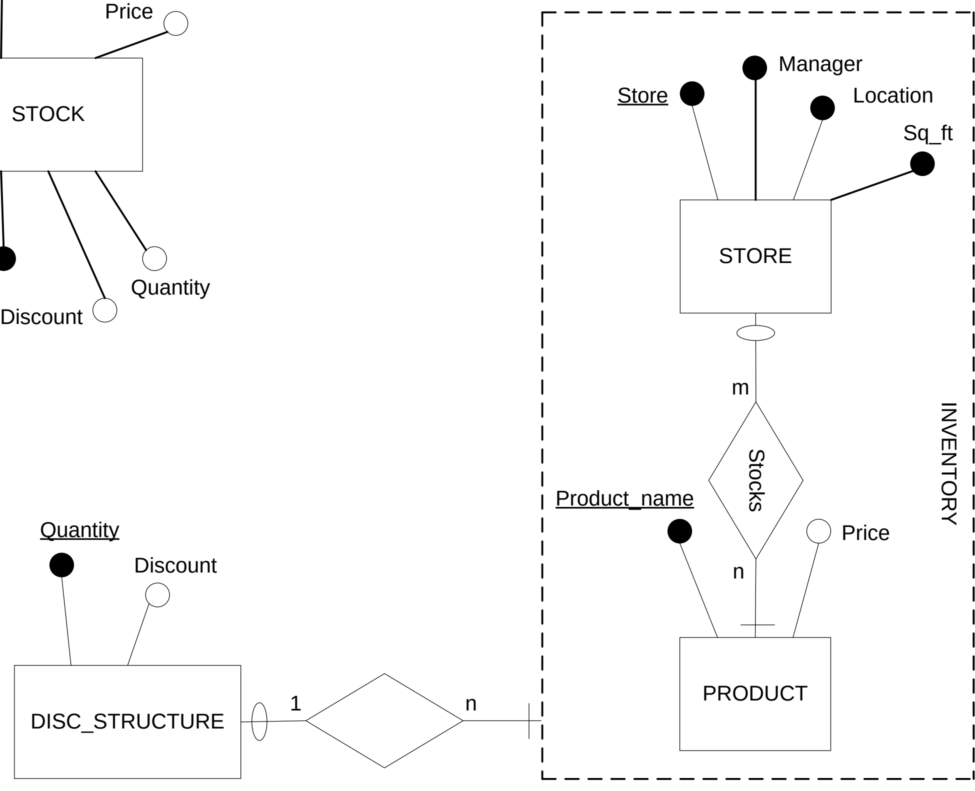
Figure 7.1b Relation schema for the entity type, STOCK

As it was . . . and As it should have been . . .

STOCK, as it was



STOCK, as it should have been



The Source of Data Redundancy/Modification Anomalies

- How do we systematically identify data redundancies?

- How do we know how to decompose the base relation schema under investigation (e.g., STOCK)?

- How do we know that the decomposition is correct and complete (without looking at sample data)?

Undesirable function dependencies are the 'seeds" of data redundancy leading to modification anomalies.

Whence Functional Dependencies?!

- Abbreviated Form: FD

- Functional dependencies are essentially technical translations of user-specified business rules expressed as constraints in a relation schema, so they cannot be ignored or discarded when undesirable.

- Functional dependency is the building block of “normalization” principles.

What is a Functional Dependency (FD) ?

- Specificity of relationship between attributes in a relation schema

- Definition: An attribute A (atomic or composite) in a relation schema R functionally determines another attribute B (atomic or composite) in R if:
  - for a given value a_1 of A there is a *single, specific* value b_1 of B in every relation state r_i of R.

  - Expressed as A → B, where

    A is called the determinant and

    B is referred to as the dependent

Examples of Functional Dependency

In the STOCK relation

- Store → Location;
- Store → Sq_ft;
- Store → Manager;
- Product → Price
- {Store, Product} → Quantity;
- Quantity → Discount

Examples of FDs in the Relational Instance STOCK

STOCK

Store	Product	Price	Quantity	Location	Discount	Sq_ft	Manager
15	Refrigerator	1850	120	Houston	5%	2300	Metzger
15	Dishwasher	600	150	Houston	5%	2300	Metzger
13	Dishwasher	600	180	Tulsa	10%	1700	Metzger
14	Refrigerator	1850	150	Tulsa	5%	1900	Schott
14	Television	1400	280	Tulsa	10%	1900	Schott
14	Humidifier	55	30	Tulsa		1900	Schott
17	Television	1400	10	Memphis		2300	Creech
17	Vacuum Cleaner	300	150	Memphis	5%	2300	Creech
17	Dishwasher	600	150	Memphis	5%	2300	Creech
11	Computer		180	Houston	10%	2300	Creech
11	Refrigerator	1850	120	Houston	5%	2300	Creech
11	Lawn Mower	300		Houston		2300	Creech

Product → Price

because all tuples of STOCK have the same Price value for any given Product value

Likewise, **Store → Location, Quantity → Discount** and **{Store, Product} → Quantity**

However, **Location -|-> Store** or **Price -|-> Product**

More on FD . . .

- Undesirable function dependencies are the ‘seeds” of data redundancy leading to modification anomalies.

- An FD in R is “undesirable” when the determinant in that FD is not a candidate key of R.

Trivial Dependency

A $\rightarrow$ B in relation R is a trivial dependency

– if and only if B is a subset of A.

– Example:

{Store, Product} → Store;

{Store, Product} → Product;

– trivial dependencies do not provide any additional information

(i.e., do not add any new constraints on R)

Note that…

- An FD is a property of the semantics (i.e., meaning) of the relationship among attributes in a relation schema emerging from the business rules.

- An FD is a property of the relation schema R, not of particular relation state r of R. Therefore, an FD cannot be automatically inferred from <u>any</u> relation state r of R.

- An FD must be explicitly specified as a constraint and the source for this specification is the business rules of the application domain.

F and Closure of F (F+)

- The set of *semantically obvious* FDs specified on a relation schema R is denoted as F.

- The set that includes F and all other FDs inferred from F is called the closure of F, often denoted as F+.

- Having specified F from the semantics of the attributes of a relation schema R, the designer can develop the closure of F (i.e., F+).

- A set of inference rules about functional dependencies developed by Armstrong is referred to as the Armstrong axioms and are useful in deriving F+.

Example

Given a set of semantically obvious FDs, F{fd1, fd2} where

- fd1: {Store, Product} → Quantity
- fd2: Quantity → Discount

One can infer the presence of

- fd3: {Store, Product} → Discount   in F+

Note: Since trivial dependencies do not provide any additional information, they are usually excluded from F+.

Armstrong's Axioms

Table 7.1 Inference rules for functional dependencies: Armstrong's axioms

Rule	Definition
Reflexivity	If Y is a subset of X [i.e., if X is (A,B,C,D) and Y is (A,C)], then X → Y. (The reflexivity rule defines trivial dependency as a dependency that is impossible to *not* satisfy.)
Augmentation	If X → Y, then {X,Z} → {Y,Z}; also, {X,Z} → Y.
Transitivity	If X → Y, and Y → Z, then X → Z.
Decomposition	If X → {Y,Z}, then X → Y and X → Z.
Union (or additive)	If X → Y, and X → Z, then X → {Y,Z}.
Composition	If X → Y, and Z → W, then {X,Z} → {Y,W}.
Pseudotransitivity	If X → Y, and {Y,W} → Z, then {X,W} → Z.

References: Armstrong, W. W. "Dependence Structures of Data Base Relationships" *Proc. IFIP Congress*, Stockholm, Sweden (1974); Darwen, H. "The Role of Functional Dependencies in Query Decomposition," In C.J. Date and H. Darwen, *Relational Database Writings 1989 – 1991*, Addison-Wesley (1992).

Cover of F Defined

- A set of FDs **G** is a cover for another set of FDs F if and only if:

  G+ = F+

- **G** ≡ F means
  - **G** and F are equivalent
  - G is a cover for F
  - F is a cover for G

Minimal (Canonical) Cover (Gc) Defined

- **Gc** is a subset of G such that
  - no FD in **Gc** is redundant .
  - That is, no FD from **Gc** can be discarded without rendering **Gc** into a set not equivalent to **G** and therefore not equivalent to F.
  - **Gc** is called the minimal (or canonical) cover of G and F.

Properties of Minimal Cover (Gc)

- The *dependent* (right side) in every FD in **Gc** is a singleton attribute.
  - This is known as the *standard or canonical form* of an FD and is intended to simplify the conditions and algorithms that ensure absence of redundancies in **Gc**.
- The *determinant* (left side) of every FD in **Gc** should be irreducible.
  - That is, no attribute can be discarded from the determinant of any FD without rendering **Gc** into some set not equivalent to **Gc**.

An Algorithm to Compute the Minimal Cover (Gc) for F

i. Set G to F.

ii. Convert all FDs in G to *standard (canonical) form* – i.e., the right side (dependent attribute) of every FD in G should be a singleton attribute.

iii. Remove all redundant attributes from the left side (determinant) of the FDs in G.

iv. Remove all redundant FDs from G.

An Algorithm to Compute Gc (continued)

Two conditions in the algorithm are noteworthy:

- The execution of this algorithm may yield different results depending on the order in which the candidates for removal (both attributes and FDs) are evaluated
  - this confirms the fact that multiple minimal covers for F are possible
- Caution: Steps iii and iv of the algorithm are not interchangeable. Executing step iv before step iii will not always return a minimal cover.

Derivation of Minimal Cover: An Exercise

Consider the relation schema

R {Student, Advisor, Subject, Grade}

and a set of FDs F [fd1, fd2, fd3] that prevails over R where:

- fd1: {Student, Advisor} $\rightarrow$ {Grade, Subject};
- fd2: Advisor $\rightarrow$ Subject;
- fd3: {Student, Subject} $\rightarrow$ {Grade, Advisor}

Derive the minimal cover for F.

Derivation of Minimal Cover: An Exercise (continued)

F in standard form:
- fd1a: {Student, Advisor} → Grade;
- fd1b: {Student, Advisor} → Subject;
- fd2: Advisor → Subject;
- fd3a: {Student, Subject} → Grade;
- fd3b: {Student, Subject} → Advisor

Minimal cover F_c of F:
- fd2: Advisor → Subject;
- fd3a: {Student, Subject} → Grade;
- fd3b: {Student, Subject} → Advisor

<u>Note</u>: fd1a and fd2b are redundant FDs since they can be derived from the other three FDs.

Closure of a Set of Attributes

Given a relation schema, R, a set of FDs, F, that holds in R, and a subset Z of attributes of R:

- *The closure Z^+ of Z under F is the set of attributes of R functionally dependent on Z.*

A Simple Illustration

Given R (A, B, C, D) and F [fd1, fd2, fd3] where

fd1: B $\rightarrow$ {G, H}; fd2: A $\rightarrow$ B; fd3: C $\rightarrow$ D

- *A^+ = Closure [A | F] = {A, B, G, H}*

Note: Attributes C and D are not in A^+.

An algorithm to compute attribute closure: See Section 7.2.4

Derivation of the First Candidate Key of R | F (The Synthesis Approach)

Given a relation schema, R, and a set of FDs, F that holds in R:

– *Find a subset Z of attributes of R such that the Z^+, closure [Z | F] includes all attributes of R*

<u>A Simple Illustration</u>

Given R (A, B, C, D) and F [fd1, fd2, fd3] where

fd1: B → {G, H}; fd2: A → B; fd3: C → D

What is a candidate key of R?

A^+, Closure [A | F] = {A, B, G, H}

C^+, Closure [C | F] = {C, D}

$\{A, C\}^+$, Closure [{A, C} | F] = {A, B, G, H, C, D}

Thus, {A, C} is a candidate key of R | F.

Derivation of the First Candidate Key of R | F (The Decomposition Approach)

Given the universal relation schema R {A1, A2, A3, . . . , An}

- Step 1: Set superkey, K of R = {A1, A2, A3, . . . , An}

- Step 2: Remove an attribute Ai, (i = 1, 2, 3, ..., n) from R such that {K – Ai} is still a superkey, K′, of R

  Note: *In order for K' to be a superkey of R, the FD: (K' → Ai) should persist in* F^+

- Step 3: Repeat step 2 above recursively until K′ is further irreducible

The irreducible K′ is a candidate key of R under the set of FDs, F.

Derivation of the First Candidate Key of R | F (The Decomposition Approach) (continued)

A universal relation schema that includes all functional dependencies is:

URS (A, B, C, D, G, H)

Step 1. Set superkey, K, of URS = {A, B, C, D, G, H}

K = {A, B, C, D, G, H}

Step 2. Remove attribute H from the URS; does (K' → H) persist in F^+

where K' = {A, B, C, D, G}

Answer: Yes, K' → H since B → H

Step 3. Remove attribute G from the URS; does (K' → G) persist in F+?

where K' = {A, B, C, D}

Answer: Yes, K' → G since B → G

Step 4. Remove attribute D from the URS; does (K' → D) persist in F+?

where K' = {A, B, C}

Answer: Yes, K' → D since C → D

Derivation of the First Candidate Key of R | F (The Decomposition Approach) (continued)

Step 5. Remove attribute C from the URS; does (K' → C) persist in F+?
where K' = {A, B}
Answer: No; K' → C does not persist in F+ since {A, B} not → C
So, C cannot be removed from the current K'

Step 6. Remove attribute B from the URS; does (K' → B) persist in F+?
where K' = {A, C}
Answer: Yes, K' → B since A → B

Step 7. Remove attribute A from the URS; does (K' → A) persist in F+?
where K' = {C}
Answer: No; K' → A does not persist in F+ since C not → A
So, A cannot be removed from the current K'

At this point, it can be seen that K' = {A, C} is a superkey that cannot be further reduced and thus becomes a candidate key of URS.

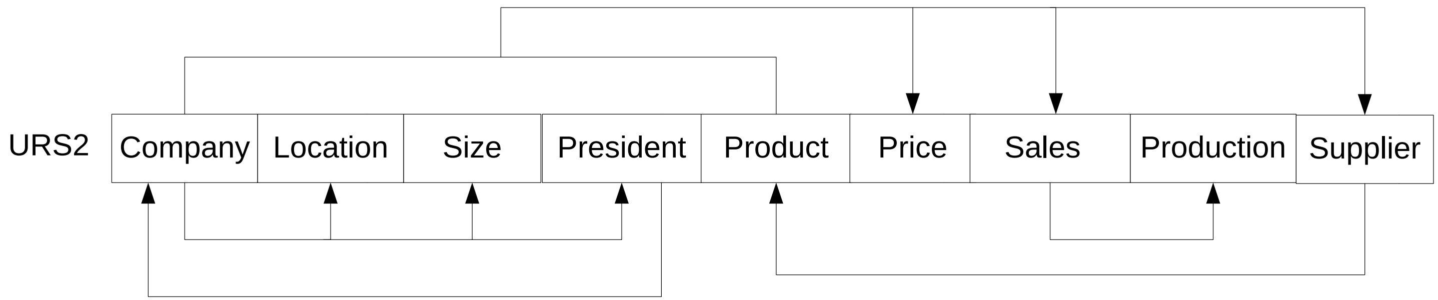
Derivation of Other Candidate Keys of R | F

- If Fc contains an FD, fdx, where a candidate key of R is a dependent, then the determinant of fdx is also a candidate key of R.
- When a candidate key of R is a composite attribute, for each key attribute (atomic or composite), evaluate if the key attribute is a dependent in an FD, fdy, in Fc. If so, then the determinant of fdy, by the rule of pseudotransitivity, can replace the key attribute under consideration, thus yielding additional candidate key(s) of R.
- Repetition of the above two steps for every candidate key of R will systematically reveal all the other candidate key(s), if any, of R.

Choosing the Primary Key from Among the Candidate Keys

- While we have noted that the choice of primary key from among the candidate keys is essentially arbitrary, some rules of thumb are often helpful in this regard:
  - *A candidate key with the least number of attributes may be a good choice.*
  - *A candidate key whose attributes are numeric and/or of small sizes may be easy to work with from a developer's perspective.*
  - *A candidate key that is a determinant in a functional dependency in F rather than F+ may be a good choice because it is probably <u>semantically obvious</u> from the user's perspective.*
  - *Surrogate keys should only be used as a last resort.*

Example



‘**Candidate’ Keys**:

{Company, Product}; {Company, Supplier};
{President, Product}; {President, Supplier}

‘**Chosen’ Primary Key**:

{Company, Product}

Key Versus Non-Key Attributes

- An attribute, atomic or composite, in a relation schema, R, is called a *key attribute* if it is a <u>proper subset</u> of <u>any</u> candidate key of R.

- Any attribute, atomic or composite, that is not a member (<u>not a subset)</u> of any candidate key is a non-key attribute

- A candidate key of R is neither a key attribute nor a non-key attribute of R.

- Based on the above discussion, we have an alternative definition for a candidate key from this point forward:
  A candidate key of a relation schema R fully functionally determines all attributes of R.

Prime Versus Non-prime Attributes

- Any attribute, atomic or composite, in a relation schema R that is a proper subset of the primary key of R is called a *prime attribute*.

- An attribute of R that is not a member (not a subset) of the primary key is *non-prime attribute* except when it is a candidate key of R.

- Any candidate key of R not chosen as the primary key is referred to as an alternate key of R and like the primary key, is neither a prime nor a non-prime attribute of R.

Prime Versus Non-prime Attributes (continued)

Table 7.4 Attribute roles in URS2

	Role of the Attribute	
Attribute	**Key/Non-key Attribute**	**Prime/Non-prime Attribute**
Company	Key attribute	Prime attribute
Location	Non-key attribute	Non-prime attribute
Size	Non-key attribute	Non-prime attribute
President	Key attribute	Non-prime attribute
Product	Key Attribute	Prime attribute
{Company, Product}	*Candidate key*	*Primary key*
{President, Product}	*Candidate key*	*Alternate key*
Price	Non-key attribute	Non-prime attribute
Sales	Non-key attribute	Non-prime attribute
Production	Non-key attribute	Non-prime attribute
Supplier	Key attribute	Non-prime attribute
{Company, Supplier}	*Candidate key*	*Alternate key*
{President, Supplier}	*Candidate key*	*Alternate key*
Note: Any composite attribute that includes one or more non-key attribute(s) is a non-key attribute.		