

CGS 2545: Database Concepts Spring 2012

Chapter 4 – Logical Database Design And The Relational Data Model – Part 2

Instructor : Dr. Mark Llewellyn
markl@cs.ucf.edu
HEC 236, 407-823-2790
<http://www.cs.ucf.edu/courses/cgs2545/spr2012>

Department of Electrical Engineering and Computer Science
Computer Science Division
University of Central Florida



Introduction To Normalization

- In general, the goal of a relational database design is to generate a set of relation schemas that create an accurate representation of the real-world situation that is being modeled.
 - The design must also allow information to be stored without unnecessary redundancy, yet also allow for that information to be retrieved efficiently.
- A technique that can be used to identify this set of suitable relational schemas is called **normalization**.
- The process of normalization builds a set of schemas, each of which is in an appropriate *normal form*.
- Normalization is a bottom-up approach to database design that begins by examining the relationships between attributes.
- To determine if a relation schema is in one of the desirable normal forms, additional information is required about the real-world scenario that is being modeled. Most of this additional information is represented by a type of data dependency known as a **functional dependency**.



Introduction To Normalization

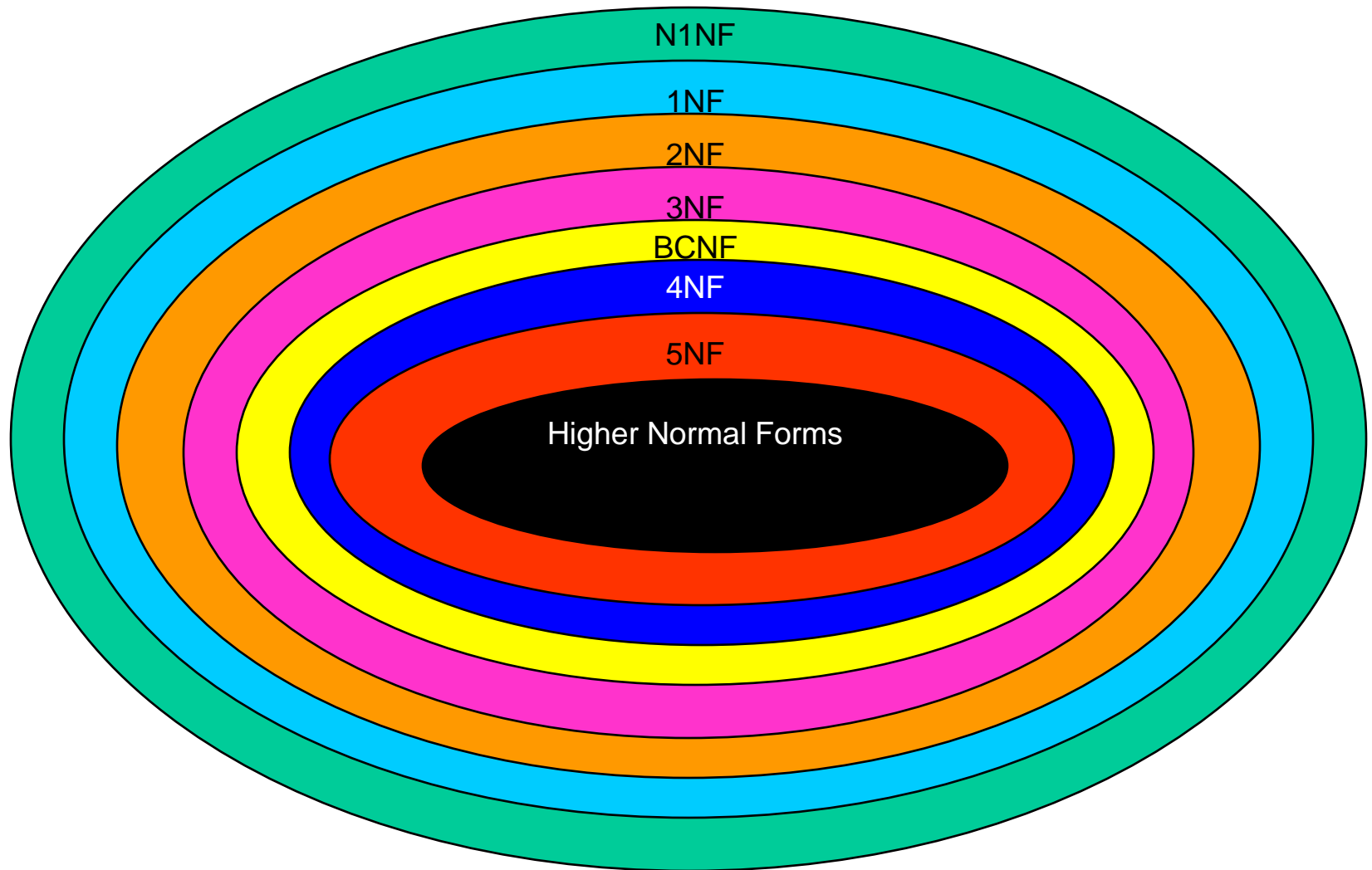
- The process of normalization can be defined formally as:

Normalization: A technique for producing a set of relational schemas with desirable properties given the data requirements pertaining to the real-world situation that is being modeled.

- The process of normalization was first developed in the early 1970s by E.F. Codd.
- Normalization is most often performed as a series of tests on a relational schema to determine whether it satisfies or violates the requirements of a given normal form.
- Codd initially proposed three normal forms called first (1NF), second (2NF), and third (3NF). Subsequently, R. Boyce and Codd together introduced a stronger definition for third normal form called Boyce-Codd Normal Form (BCNF).
- All four of these normal forms are based upon the concept of a functional dependency. Higher normal forms that go beyond BCNF, such as fourth (4NF) and fifth (5NF), as well as several others, have also subsequently been introduced. These higher normal forms utilize other types of data dependencies and some of these apply to situations that are quite rare. We will concentrate only on the first four normal forms and not examine any of the higher normal forms.



Relationship Between Normal Forms



Introduction To Normalization

- The process of normalization is a formal method that identifies relational schemas based upon their primary or candidate keys and the functional dependencies that exists amongst their attributes.
- Normalization is primarily a tool to validate and improve a logical design so that it satisfies certain constraints that *avoid unnecessary duplication of data*.
- Normalization is the process of decomposing relations with anomalies to produce smaller, *well-structured* relations.



Introduction To Normalization

- A well-structured relation contains minimal data redundancy and allows users to insert, delete, and update rows without causing data inconsistencies.
- Goal is to avoid anomalies
 - **Insertion Anomaly** – adding new rows forces user to create duplicate data.
 - **Deletion Anomaly** – deleting rows may cause a loss of data that would be needed for other future rows.
 - **Modification Anomaly** – changing data in a row forces changes to other rows because of duplication.



Example – Anomalies In A Relation

EMPLOYEE2

<u>Emp_ID</u>	Name	Dept_Name	Salary	<u>Course_Title</u>	Date_Completed
100	Margaret Simpson	Marketing	48,000	SPSS	6/19/200X
100	Margaret Simpson	Marketing	48,000	Surveys	10/7/200X
140	Alan Beeton	Accounting	52,000	Tax Acc	12/8/200X
110	Chris Lucero	Info Systems	43,000	SPSS	1/12/200X
110	Chris Lucero	Info Systems	43,000	C++	4/22/200X
190	Lorenzo Davis	Finance	55,000		
150	Susan Martin	Marketing	42,000	SPSS	6/19/200X
150	Susan Martin	Marketing	42,000	Java	8/12/200X

Question – Is this a relation?

Answer – Yes: unique rows and no multivalued attributes

Question – What's the primary key?

Answer – Composite: Emp_ID,
Course_Title



Anomalies in this Table

- **Insertion** – can't enter a new employee without having the employee take a class.
- **Deletion** – if we remove employee 140, we lose information about the existence of a Tax Acc class.
- **Modification** – giving a salary increase to employee 100 forces us to update multiple records.

Why do these anomalies exist?

Because there are two themes (entity types) into one relation. This results in duplication, and an unnecessary dependency between the entities

General rule of thumb: a table should not pertain to more than one entity type



Brief Overview Of The Steps in Normalization

- **First Normal Form (1NF):** All multi-valued attributes have been removed from the table. Only a single value (possibly null) exists at the intersection of each row and column of the table.
- **Second Normal Form (2NF):** All partial functional dependencies have been removed. [Non-key attributes are identified by only the full primary key.]
- **Third Normal Form (3NF):** All transitive functional dependencies have been removed. [Non-key attributes are identified by only the primary key.]
- **Boyce-Codd Normal Form (BCNF):** Any remaining anomalies that result from functional dependencies have been removed. [More than one primary key existed for the same non-key attributes.]



Brief Overview Of The Steps in Normalization

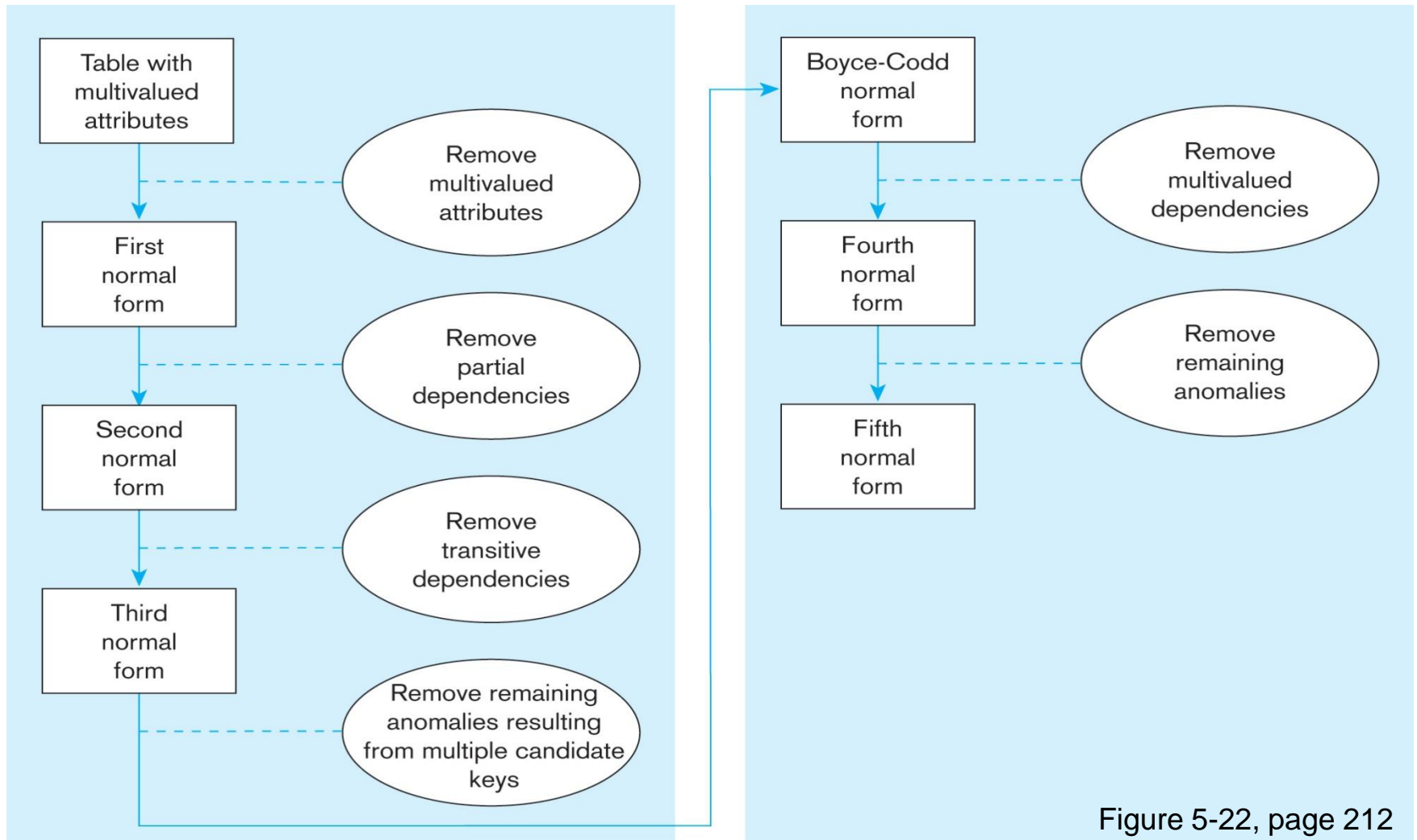


Figure 5-22, page 212

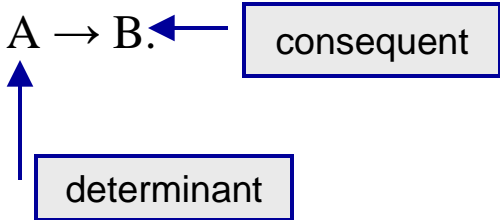


Important Note

- The design of a relational database should have included a conceptual modeling step (producing an ER diagram) for the enterprise (as we have done).
- This step was followed by a transformation process that converted the ER diagram into a set of relational tables.
- The first step in the transformation process generated a table (relation) for every multi-valued attribute for a given entity.
- This means that every table (relation) that was created was in fact a relation and thus is in 1NF.
- In our earlier discussion of anomalies, the table was in 1NF but was not a well-structured table as it contained certain anomalies. Normalization will remove these anomalies.



Functional Dependencies

- A **functional dependency** is a constraint between two attributes (or sets of attributes).
 - For any relation R, attribute B is functionally dependent on attribute A if, for every valid instance of A, that value of A uniquely determines the value of B.
 - The functional dependency of B on A is denoted as: $A \rightarrow B$.

- Example:

EMP_COURSE (Emp_ID, Course_Title, Date_Completed)

The relation instance shown on the right satisfies the functional dependency

$\text{Emp_ID, Course_Title} \rightarrow \text{Date_Completed}$

Emp_ID	Course_Title	Date_Completed
100	Excel	4/1/2006
100	Access	5/20/2005
140	Tax Acct.	3/14/2000
110	Visual Basic	6/6/2006
110	C++	11/16/2004
150	Excel	6/27/2003
150	Access	8/12/2002



A 1NF, But Not Well-structured, Table

<u>Order_ID</u>	Order_ Date	Customer_ ID	Customer_ Name	Customer_ Address	<u>Product_ID</u>	Product_ Description	Product_ Finish	Unit_ Price	Ordered_ Quantity
1006	10/24/2004	2	Value Furniture	Plano, TX	7	Dining Table	Natural Ash	800.00	2
1006	10/24/2004	2	Value Furniture	Plano, TX	5	Writer's Desk	Cherry	325.00	2
1006	10/24/2004	2	Value Furniture	Plano, TX	4	Entertainment Center	Natural Maple	650.00	1
1007	10/25/2004	6	Furniture Gallery	Boulder, CO	11	4-Dr Dresser	Oak	500.00	4
1007	10/25/2004	6	Furniture Gallery	Boulder, CO	4	Entertainment Center	Natural Maple	650.00	3



Anomalies in this Table

- **Insertion** – if new product is ordered for order 1007 of existing customer, customer data must be re-entered, causing duplication.
- **Deletion** – if we delete the Dining Table from Order 1006, we lose information concerning this item's finish and price.
- **Update** – changing the price of product ID 4 requires update in several records.



Functional Dependencies in this Table

Full Dependency

Transitive Dependencies

<u>Order_ID</u>	Order_Date	Customer_ID	Customer_Name	Customer_Address	<u>Product_ID</u>	Product_Description	Product_Finish	Unit_Price	Ordered_Quantity
-----------------	------------	-------------	---------------	------------------	-------------------	---------------------	----------------	------------	------------------

Partial Dependencies

Partial Dependencies

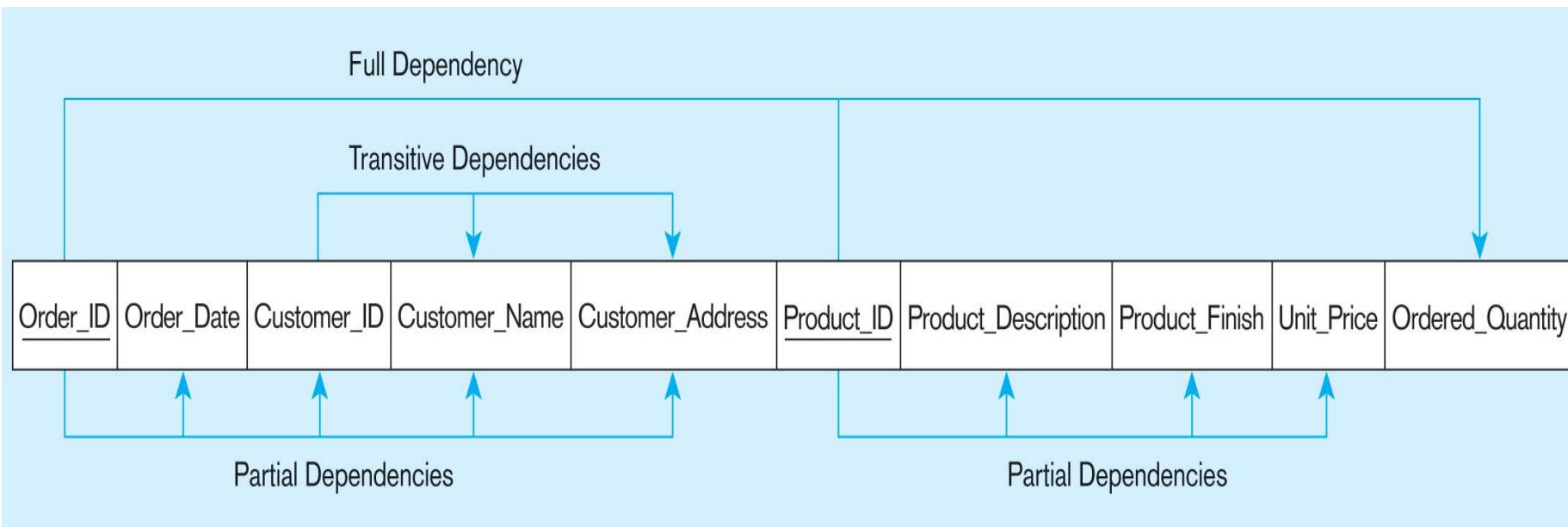


Definition of 2NF

- A relation is in 2NF if it is in 1NF and every non-key attribute is fully functionally dependent on the ENTIRE primary key.
 - Every non-key attribute must be defined by the entire key, not by only part of the key. (A partial dependency exists whenever a non-key attribute is functionally dependent on only a portion of the primary key.)
 - No partial functional dependencies exist in a 2NF relation.



Why INVOICE Table Is Not In 2NF



Order_ID → Order_Date, Customer_ID, Customer_Name, Customer_Address

Product_ID → Product_Description, Product_Finish, Unit_Price

Therefore, NOT in 2nd Normal Form



Converting A N2NF Relation Into A 2NF Relation

- To convert a relation containing partial dependencies into a 2NF relation, the following steps are required:
 1. Create a new relation for each primary key attribute (or combinations of attributes) that is a determinant in a partial dependency. That attribute is the primary key in the new relation.
 2. Move the non-key attributes that are dependent on this primary key attribute (or attributes) from the old relation into the new relation.



Converting A N2NF Relation Into A 2NF Relation

EXAMPLE

<u>Order_ID</u>	<u>Product_ID</u>	Ordered_Quantity
-----------------	-------------------	------------------

ORDER_LINE (3NF)

<u>Product_ID</u>	Product_Description	Product_Finish	Unit_Price
-------------------	---------------------	----------------	------------

PRODUCT (3NF)

<u>Order_ID</u>	Order_Date	Customer_ID	Customer_Name	Customer_Address
-----------------	------------	-------------	---------------	------------------

CUSTOMER_ORDER (2NF)



Transitive Dependencies



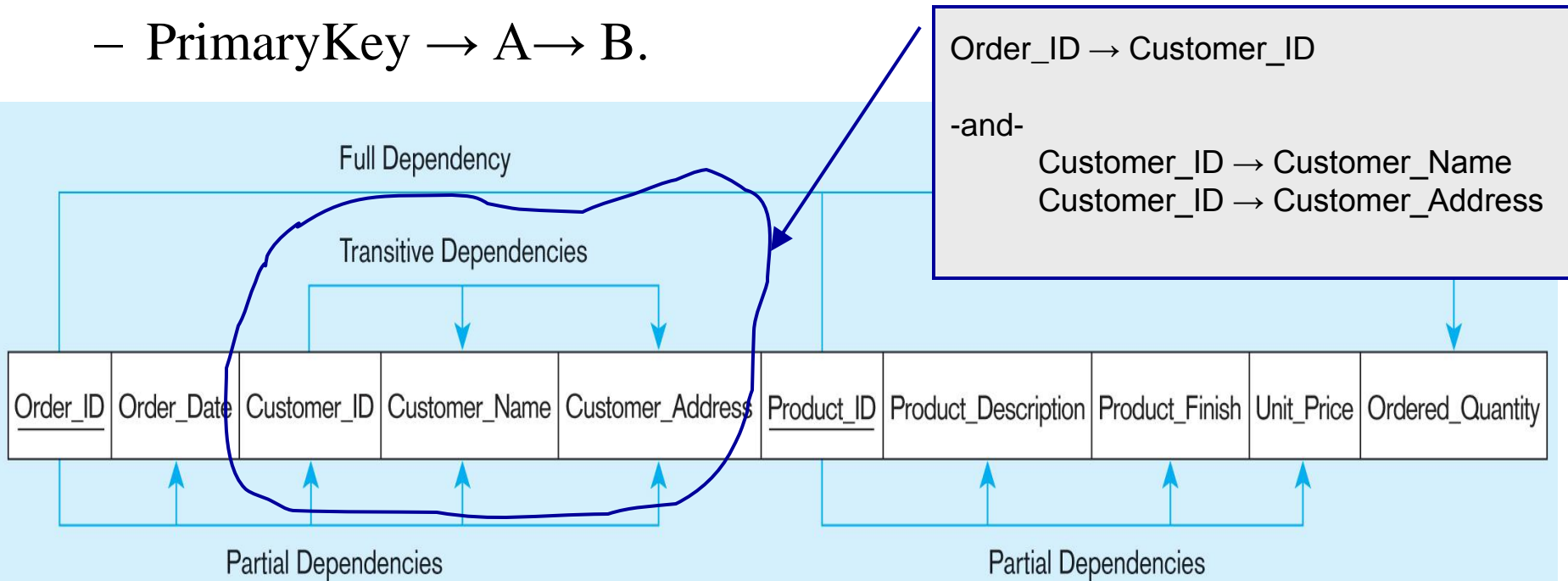
Consequences of the Definition of 2NF

- A 1NF relation will be in 2NF if any of the following conditions hold:
 1. The primary key consists of only one attribute. By definition, there cannot be a partial dependency in such a relation.
 2. No non-key attributes exists in the relation (all of the attributes in the relation are part of the primary key). By definition there are no functional dependencies (other than the trivial ones) in such a relation.
 3. Every non-key attribute is functionally dependent on the full set of primary key attributes.



Definition of 3NF

- A relation is in 3NF if it is in 2NF and **no transitive dependencies exist**.
 - A transitive dependency in a relation is a functional dependency between two (or more) non-key attributes.
 - **PrimaryKey** $\rightarrow A \rightarrow B$.



Converting A N3NF Relation Into A 3NF Relation

- To convert a relation containing transitive dependencies into a 3NF relation, the following steps are required:
 1. For each non-key attributed (or set of attributed) that is a determinant in the relation, create a new relation. That attribute (or set of attributes) becomes the primary key in the new relation.
 2. Move all of the attributes that are functionally dependent on the attribute from the old relation into the new relation.
 3. Leave the attribute (which serves as the primary key in the new relation) in the old relation to serve as a foreign key that allows an association between the two relation.



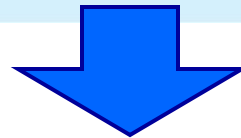
Converting A N3NF Relation Into A 3NF Relation

EXAMPLE

<u>Order_ID</u>	Order_Date	Customer_ID	Customer_Name	Customer_Address
-----------------	------------	-------------	---------------	------------------

CUSTOMER_ORDER (2NF)

Transitive Dependencies



<u>Order_ID</u>	Order_Date	<u>Customer_ID</u>
-----------------	------------	--------------------

ORDER (3NF)

<u>Customer_ID</u>	Customer_Name	Customer_Address
--------------------	---------------	------------------

CUSTOMER (3NF)



Denormalization

- Denormalization is the process of transforming normalized relations into non-normalized physical record specifications.
- In a modern computer system, the cost per unit of storage (memory) has decreased drastically in recent years, Thus, while it is still a consideration, the efficient use of storage space has become less important than in the past.
- In a modern DBMS, efficient data processing dominates the design process. In other words, speed not style takes precedence.
- Efficient processing of data is in part dependent on how close related data are maintained in memory.



Denormalization

- As we've just seen, the normalization process tends to distribute the data into many tables.
- It is often the case that all of the attributes that appear within a relation are not used together, and data from different relations are needed to be combined together to answer a query or produce a report.
- Although normalized relations solve data maintenance anomalies and minimize redundancy (and hence reduce storage space requirements), if implemented as one for one physical records, will probably not yield efficient data processing.



Denormalization

- A fully normalized database generally contains a large number of relations. For a frequently executed query that requires data from multiple, related tables, the DBMS can spend a considerable amount of time each time the query is executed in matching the related rows (a technique called **joining** that we'll see a bit later) from each relation that is required to build the query result.
- Since join operations can be quite time consuming, the processing performance differential between totally normalized and partially normalized (denormalized) databases can be quite dramatic.



Denormalization

- In general, denormalization may partition a relation into several physical records, may combine attributes from several relations together into one physical record, or a combination of both.
- There are, in general, three common types of denormalization that occur:
 - Two entities with a 1:1 relationship between them.
 - A N:M relationship (associative entity) with non-key attributes.
 - Reference data.
- We'll look more closely at each of these types on the next few pages.

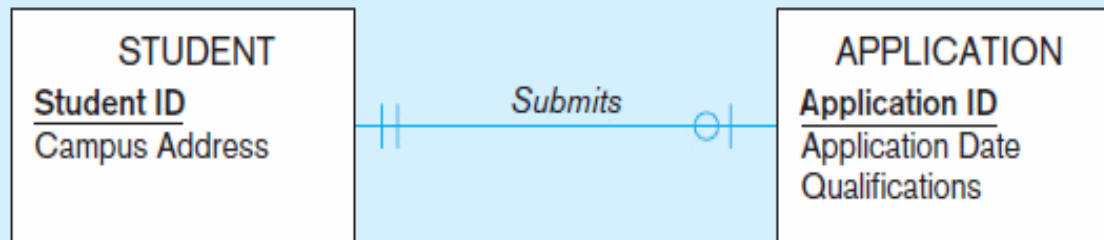


Denormalization Case: 1:1 Binary Relationship

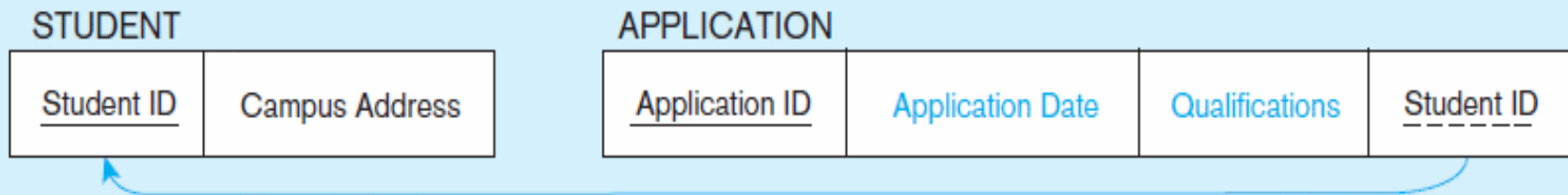
- Even in cases where one of the entities is an optional participant, if the matching entity exists most of the time, then it may be wise to combine these two relations into one record definition. This would be especially true if the access frequency between the two entity types is high).
- Consider the example shown on the next page. The ERD shows student data with optional data from a standard scholarship application a student might complete.
- In this case, one record could be formed with four fields from the Student and Application normalized relations.
- Note that in this case the attributes from the optional entity must be allowed to have null values.



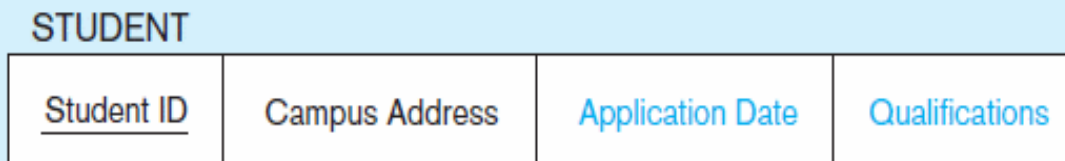
Denormalization Case: 1:1 Binary Relationship



Normalized relations:



Denormalized relation:



and Application Date and Qualifications may be null

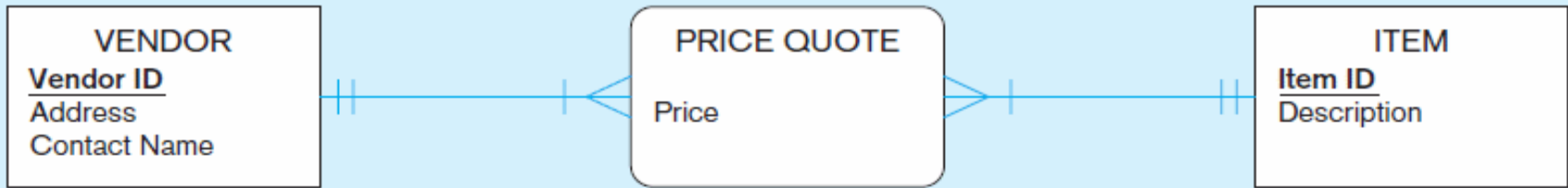


Denormalization Case: N:M (Associative Entity)

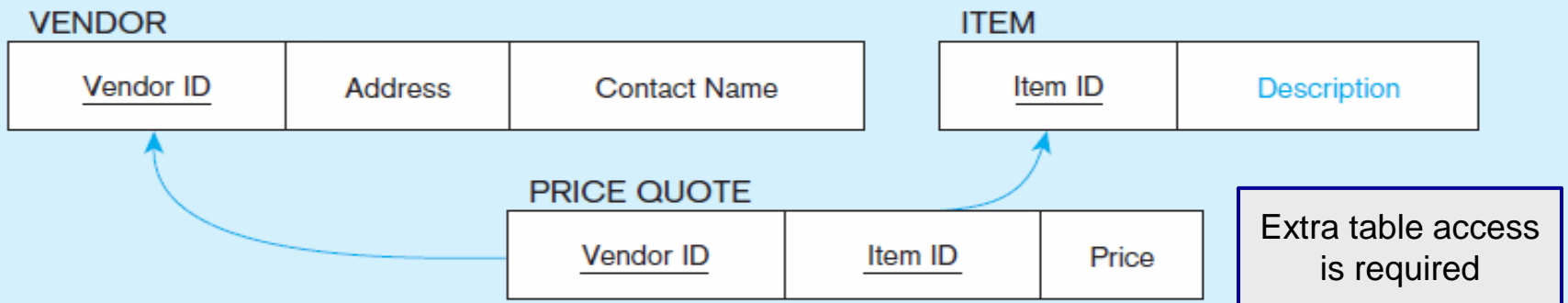
- In this case, rather than joining three files in order to extract data from the two basic entities in the relationship, it might be advisable to combine attributes from one of the entities into the record representing the N:M relationship and thus avoid one of the join operations.
- This would be most advantageous if this joining occurs frequently.
- The example on the next page illustrates this situation with price quotes for various items from different vendors. In this situation, attributes from the Item and Price Quote relations might be combined into one record to prevent the three table join operation.
- Note that this may create considerable duplication of data, since the Item attributes, such as Description, would repeat for each price quote. This would require excessive updating if duplicated data changed. Analysis of a composite usage map to study access frequencies and the number of occurrences of Price Quote per associated Vendor or Item would be essential to understand the consequences of such denormalization.



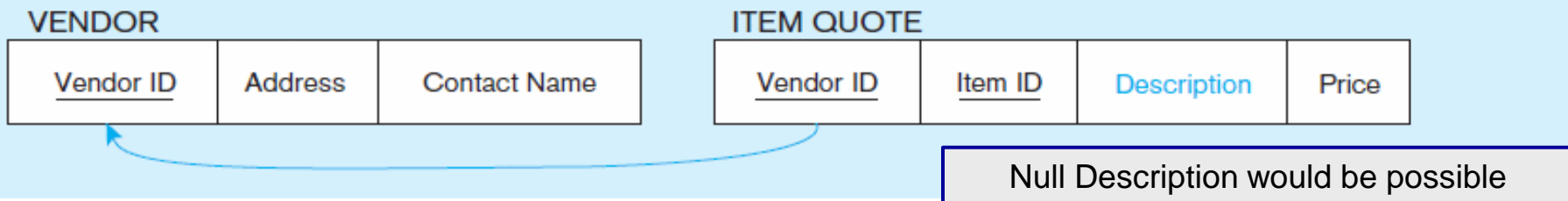
Denormalization Case: N:M (Associative Entity)



Normalized relations:



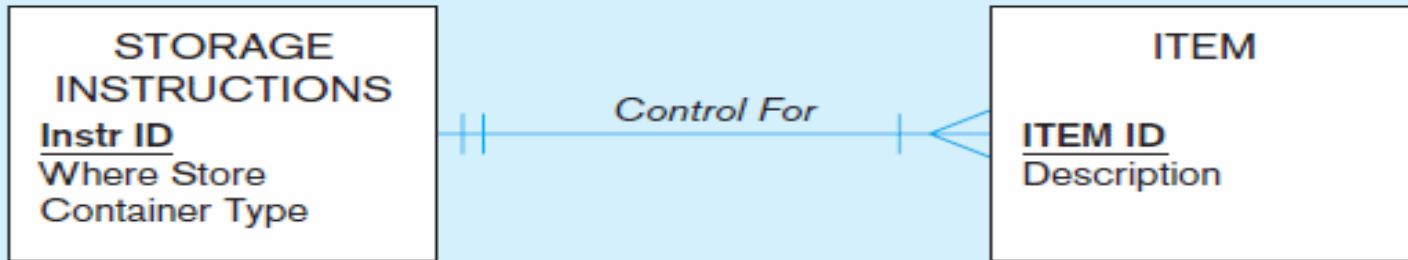
Denormalized relations:



Denormalization Case: Reference Data

- Reference data exist in an entity on one side of a 1:M relationship, and this entity participates in no other relationships.
- When this situation arises, you should seriously consider merging the two entities into one physical record definition when there are few instances of the entity on the many side for each entity instance on the one side.
- The following page illustrates this situation, in which several Items have the same Storage Instructions and Storage Instructions relates only to Items. In this case, the storage instructions could be stored in the Item record,
- Note that in so doing, redundancy and the potential for extra data maintenance will increase.





Normalized relations:

STORAGE

<u>Instr ID</u>	Where Store	Container Type
-----------------	-------------	----------------

ITEM

<u>Item ID</u>	Description	<u>Instr ID</u>
----------------	-------------	-----------------

Extra table access is required

Denormalized relation:

ITEM

<u>Item ID</u>	Description	Where Store	Container Type
----------------	-------------	-------------	----------------

Data duplication

