Chapter Four:

Database Design Using Normalization

Chapter Objectives

- To design updatable databases to store data received from another source
- To use SQL to access table structure
- To understand the advantages and disadvantages of normalization
- To understand denormalization
- To design read-only databases to store data from updateable databases

Chapter Objectives

- To recognize and be able to correct common *design problems*:
  - – The multivalue, multicolumn problem
  - – The inconsistent values problem
  - – The missing values problem
  - – The general-purpose remarks column problem

Chapter Premise

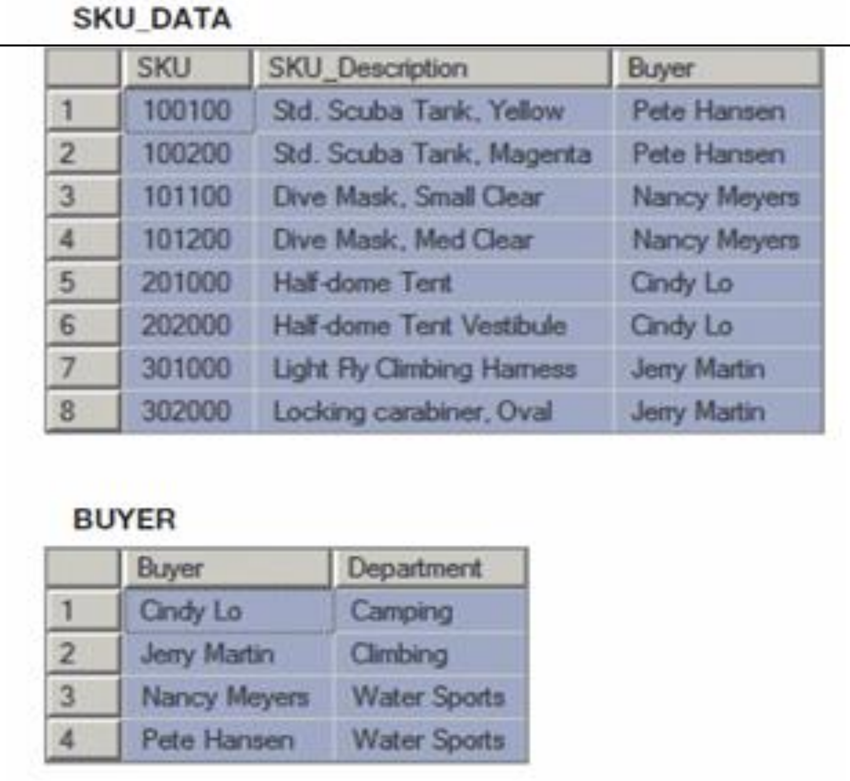
- We have received one or more tables of existing data.
- The data is to be stored in a new database.

QUESTION: Should the data be stored as received, or should it be transformed for storage?

How Many Tables?

SKU_DATA (**<u>SKU</u>**, SKU_Description, *Buyer*)
BUYER (**<u>Buyer</u>**, Department)

where SKU_DATA.*Buyer* must exist in BUYER.Buyer.



SKU_DATA

	SKU	SKU_Description	Buyer
1	100100	Std. Scuba Tank, Yellow	Pete Hansen
2	100200	Std. Scuba Tank, Magenta	Pete Hansen
3	101100	Dive Mask, Small Clear	Nancy Meyers
4	101200	Dive Mask, Med Clear	Nancy Meyers
5	201000	Half-dome Tent	Cindy Lo
6	202000	Half-dome Tent Vestibule	Cindy Lo
7	301000	Light Fly Climbing Harness	Jerry Martin
8	302000	Locking carabiner, Oval	Jerry Martin

BUYER

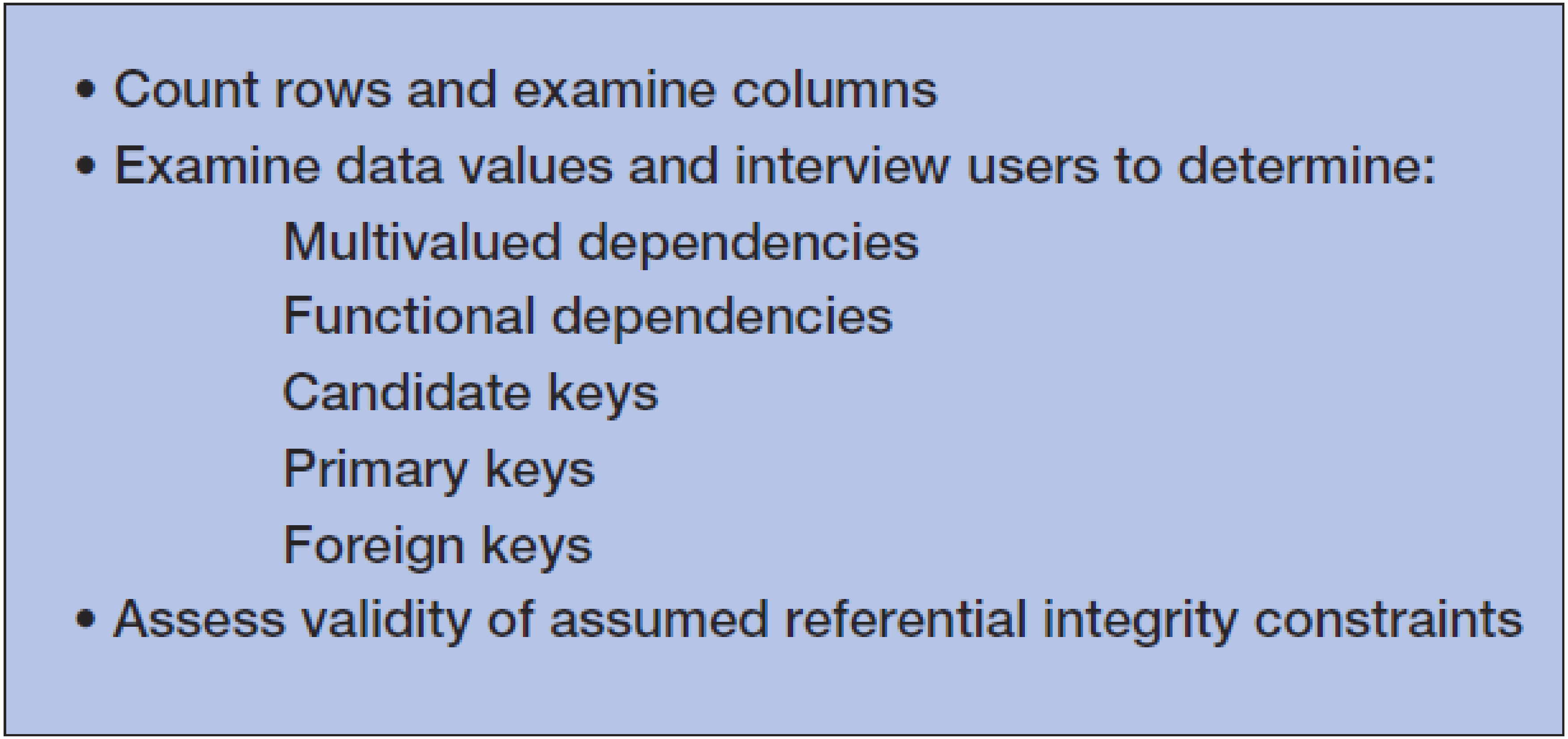
	Buyer	Department
1	Cindy Lo	Camping
2	Jerry Martin	Climbing
3	Nancy Meyers	Water Sports
4	Pete Hansen	Water Sports

Should we store these two tables as they are, or should we combine them into one table in our new database?

How well do you know your tables?

1. ACCESSING TABLE STRUCTURE

Assessing Table Structure



- Count rows and examine columns
- Examine data values and interview users to determine:

  Multivalued dependencies

  Functional dependencies

  Candidate keys

  Primary keys

  Foreign keys
- Assess validity of assumed referential integrity constraints

Counting Rows in a Table

- To count the number of rows in a table use the **SQL COUNT(*) built-in function** :

```
SELECT   COUNT(*) AS NumRows
FROM     SKU_DATA;
```

Examining the Columns

- To determine the number and type of columns in a table, use an SQL SELECT statement.

- To limit the number of rows retrieved, use the

```
SELECT  TOP 10 *  FROM  SKU_DATA;
```

```
Select * from sku_data where rownum<6;
```

Checking Validity of Assumed Referential Integrity Constraints I

- Given two tables with an assumed foreign key constraint:

SKU_DATA (SKU, SKU_Description, *Buyer*)

BUYER (Buyer, Department)

Where SKU_DATA.Buyer must exist in BUYER.Buyer

Checking Validity of Assumed Referential Integrity Constraints II

- To find any foreign key values that violate the foreign key constraint:

```
SELECT          Buyer
FROM            SKU_DATA
WHERE           Buyer NOT IN
                (SELECT SKU_DATA.Buyer
                 FROM    SKU_DATA, BUYER
                 WHERE   SKU_DATA.BUYER = BUYER.Buyer);
```

2. TYPE OF DATABASES

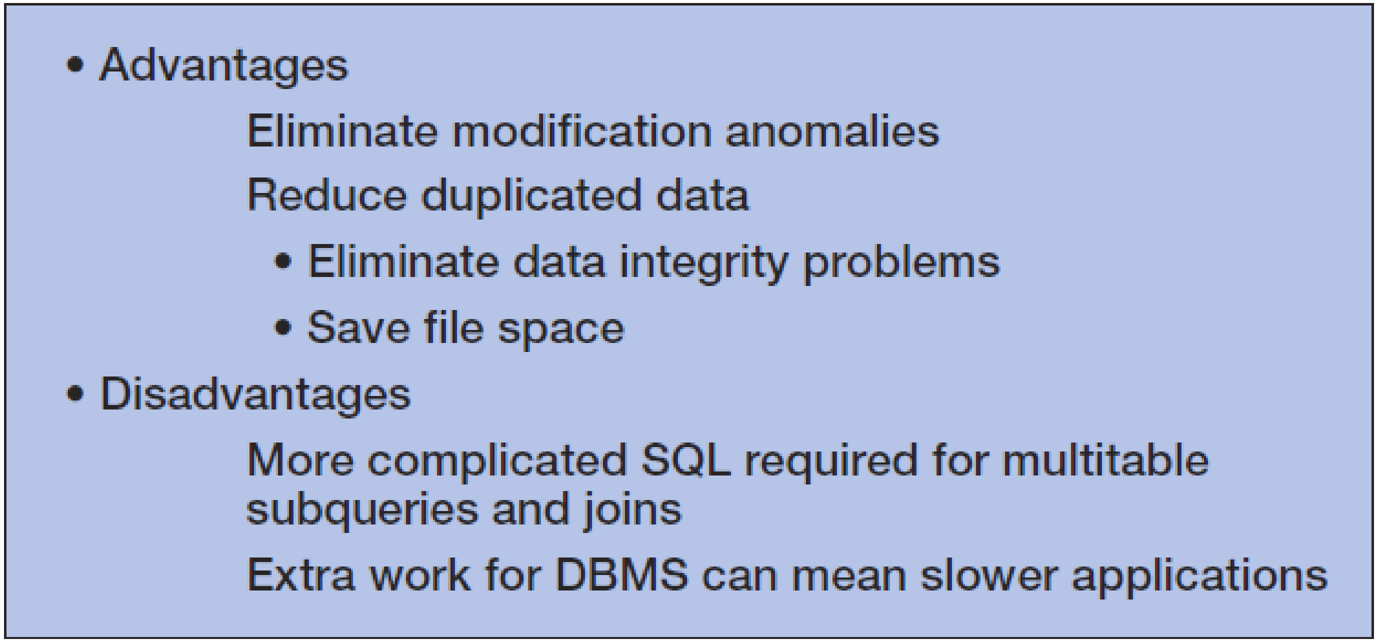
Updateable or Read-only?

- If updateable database, we normally want tables in BCNF.

- If read-only database, we may not use BCNF tables.

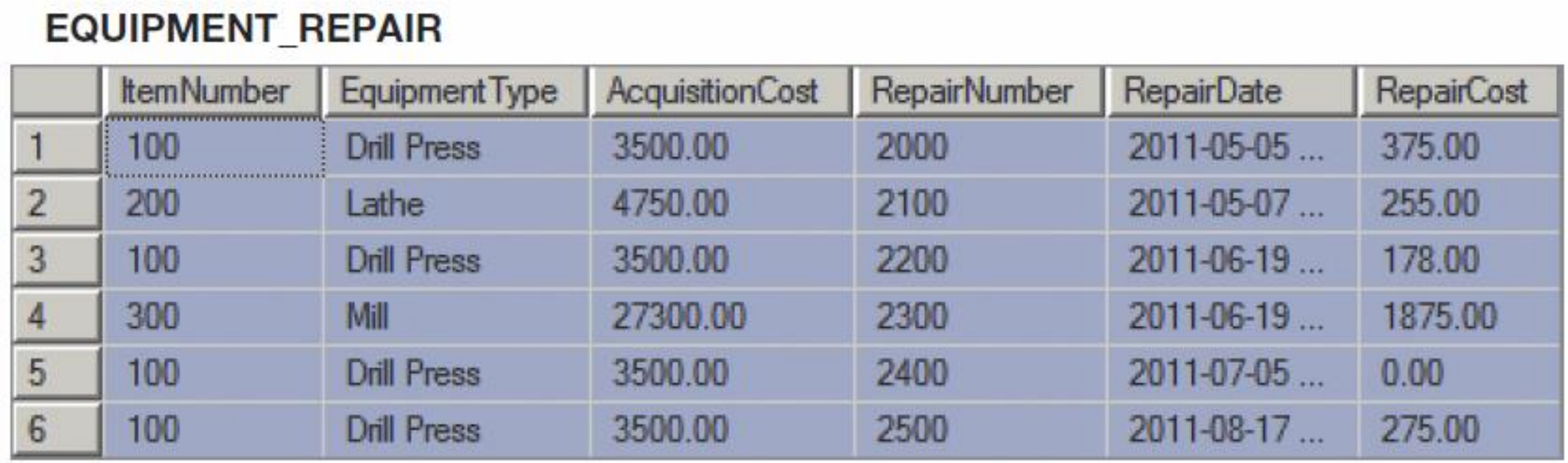
3. DESIGNING UPDATABLE DATABASES

Normalization: Advantages and Disadvantages



- Advantages
  - Eliminate modification anomalies
  - Reduce duplicated data
    - Eliminate data integrity problems
    - Save file space
- Disadvantages
  - More complicated SQL required for multitable subqueries and joins
  - Extra work for DBMS can mean slower applications

Non-Normalized Table: EQUIPMENT_REPAIR

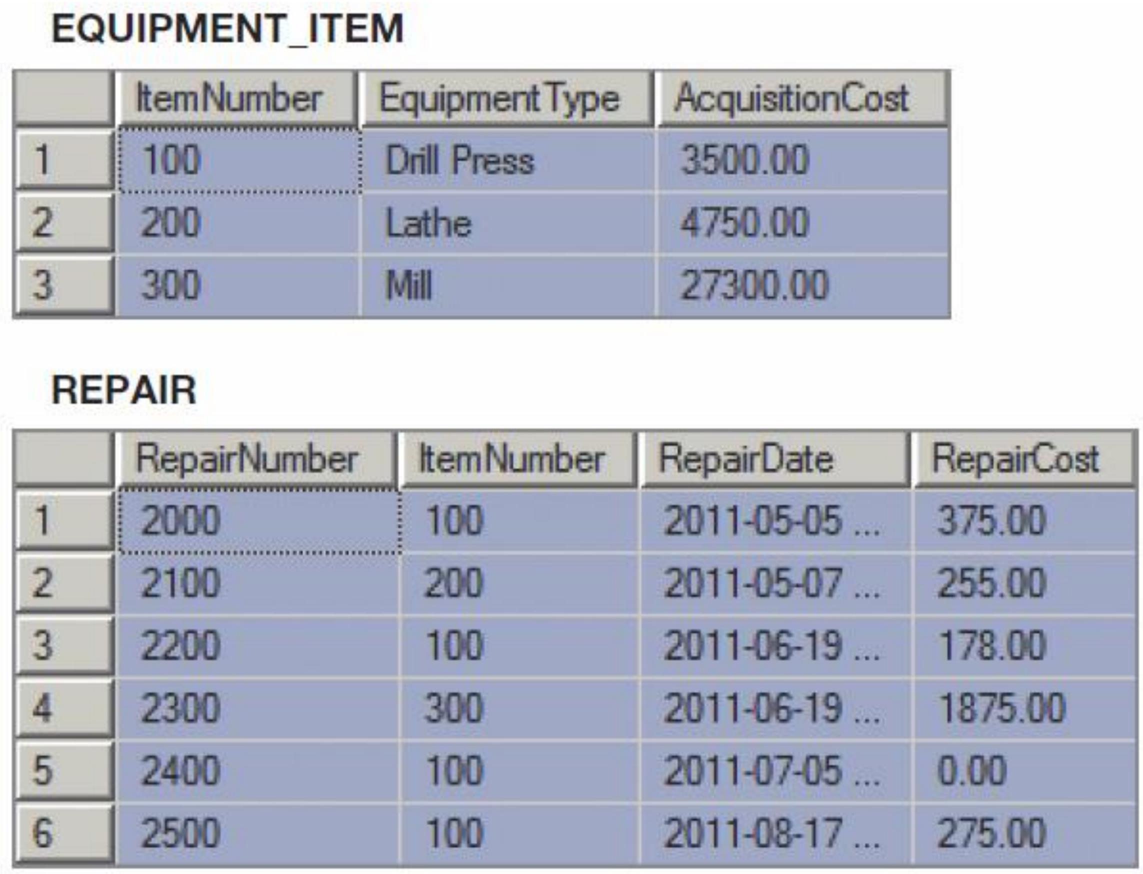


EQUIPMENT_REPAIR

	ItemNumber	EquipmentType	AcquisitionCost	RepairNumber	RepairDate	RepairCost
1	100	Drill Press	3500.00	2000	2011-05-05 ...	375.00
2	200	Lathe	4750.00	2100	2011-05-07 ...	255.00
3	100	Drill Press	3500.00	2200	2011-06-19 ...	178.00
4	300	Mill	27300.00	2300	2011-06-19 ...	1875.00
5	100	Drill Press	3500.00	2400	2011-07-05 ...	0.00
6	100	Drill Press	3500.00	2500	2011-08-17 ...	275.00

- Why?

Normalized Tables:
ITEM and REPAIR



EQUIPMENT_ITEM

	ItemNumber	EquipmentType	AcquisitionCost
1	100	Drill Press	3500.00
2	200	Lathe	4750.00
3	300	Mill	27300.00

REPAIR

	RepairNumber	ItemNumber	RepairDate	RepairCost
1	2000	100	2011-05-05 ...	375.00
2	2100	200	2011-05-07 ...	255.00
3	2200	100	2011-06-19 ...	178.00
4	2300	300	2011-06-19 ...	1875.00
5	2400	100	2011-07-05 ...	0.00
6	2500	100	2011-08-17 ...	275.00

Copying Data to New Tables

- To copy data from one table to another, use the SQL command **INSERT INTO** ***TableName*** command:

```
INSERT INTO     EQUIPMENT_ITEM
        SELECT  DISTINCT ItemNumber,  EquipmentType, AcquisitionCost
        FROM    EQUIPMENT_REPAIR;


INSERT INTO     REPAIR
        SELECT  RepairNumber, ItemNumber, RepairDate, RepairCost
        FROM    EQUIPMENT_REPAIR;
```

Choosing Not To Use BCNF

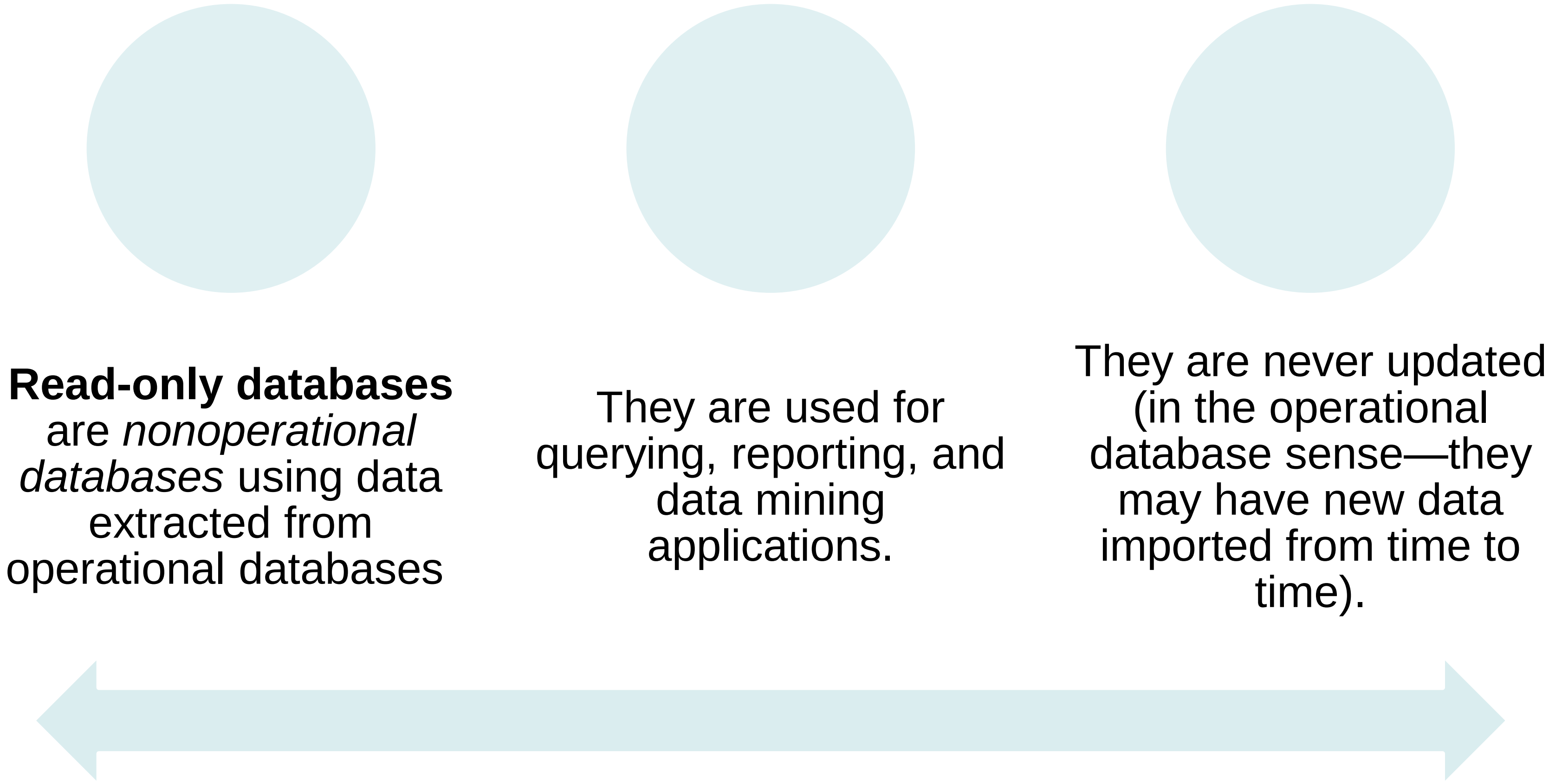
- BCNF is used to control anomalies from functional dependencies.
- *There are times when BCNF is not desirable.*
- The classic example is ZIP codes:
  - ZIP codes almost never change.
  - Any anomalies are likely to be caught by normal business practices.
  - Not having to use SQL to join data in two tables will speed up application processing.

Multivalued Dependencies

- Anomalies from multivalued dependencies are very problematic.
- ***Always*** place the columns of a multivalued dependency into a separate table (4NF).

4. DESIGNING READ-ONLY DATABASES

Read-Only Databases



Read-only databases are *nonoperational databases* using data extracted from operational databases

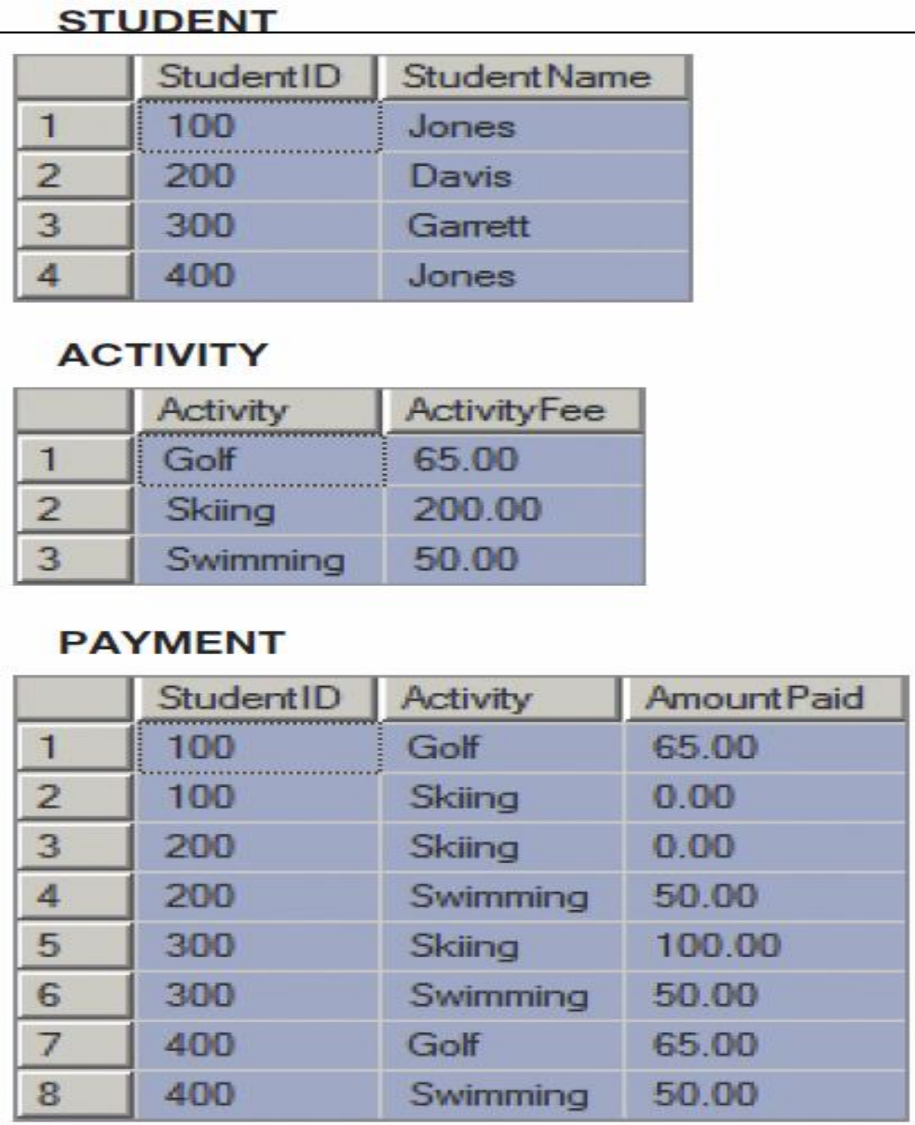
They are used for querying, reporting, and data mining applications.

They are never updated (in the operational database sense—they may have new data imported from time to time).

Denormalization

- For read-only databases, normalization is seldom an advantage.
  - Application processing speed is more important.
- **Denormalization** is the joining of the data in normalized tables prior to storing the data.
- The data is then stored in nonnormalized tables.

Normalized Tables



STUDENT

	StudentID	StudentName
1	100	Jones
2	200	Davis
3	300	Garrett
4	400	Jones

ACTIVITY

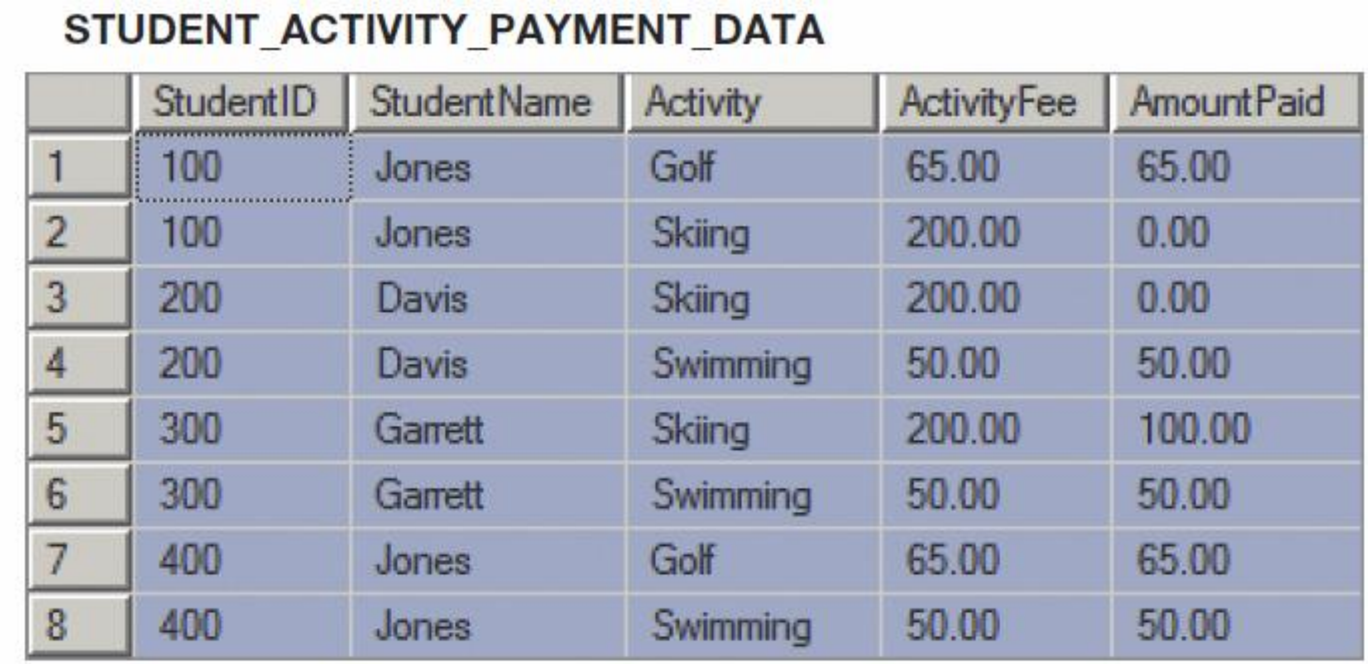
	Activity	ActivityFee
1	Golf	65.00
2	Skiing	200.00
3	Swimming	50.00

PAYMENT

	StudentID	Activity	AmountPaid
1	100	Golf	65.00
2	100	Skiing	0.00
3	200	Skiing	0.00
4	200	Swimming	50.00
5	300	Skiing	100.00
6	300	Swimming	50.00
7	400	Golf	65.00
8	400	Swimming	50.00

Denormalizing the Data

```
INSERT INTO STUDENT_ACTIVITY_PAYMENT_DATA
    SELECT      STUDENT.StudentID, StudentName, ACTIVITY.Activity, ActivityFee,
    AmountPaid
    FROM         STUDENT, PAYMENT, ACTIVITY
    WHERE       STUDENT.StudentID = PAYMENT.StudentID
        AND     PAYMENT.Activity = ACTIVITY.Activity;
```

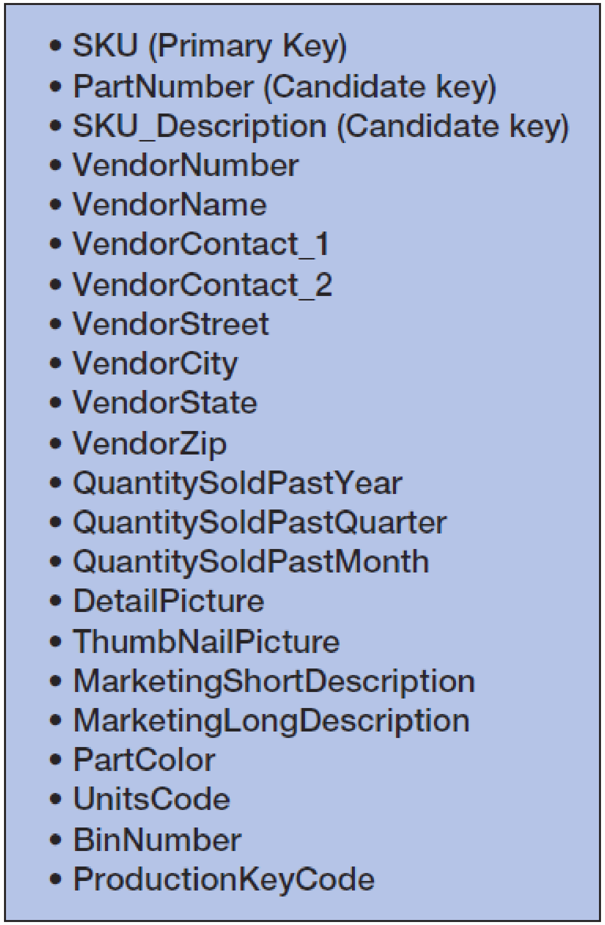


STUDENT_ACTIVITY_PAYMENT_DATA

	StudentID	StudentName	Activity	ActivityFee	AmountPaid
1	100	Jones	Golf	65.00	65.00
2	100	Jones	Skiing	200.00	0.00
3	200	Davis	Skiing	200.00	0.00
4	200	Davis	Swimming	50.00	50.00
5	300	Garrett	Skiing	200.00	100.00
6	300	Garrett	Swimming	50.00	50.00
7	400	Jones	Golf	65.00	65.00
8	400	Jones	Swimming	50.00	50.00

Customized Tables I

- Read-only databases are often designed with many copies of the same data, but with each copy customized for a specific application.
- Consider the PRODUCT table:



- SKU (Primary Key)
- PartNumber (Candidate key)
- SKU_Description (Candidate key)
- VendorNumber
- VendorName
- VendorContact_1
- VendorContact_2
- VendorStreet
- VendorCity
- VendorState
- VendorZip
- QuantitySoldPastYear
- QuantitySoldPastQuarter
- QuantitySoldPastMonth
- DetailPicture
- ThumbNailPicture
- MarketingShortDescription
- MarketingLongDescription
- PartColor
- UnitsCode
- BinNumber
- ProductionKeyCode

Customized Tables II

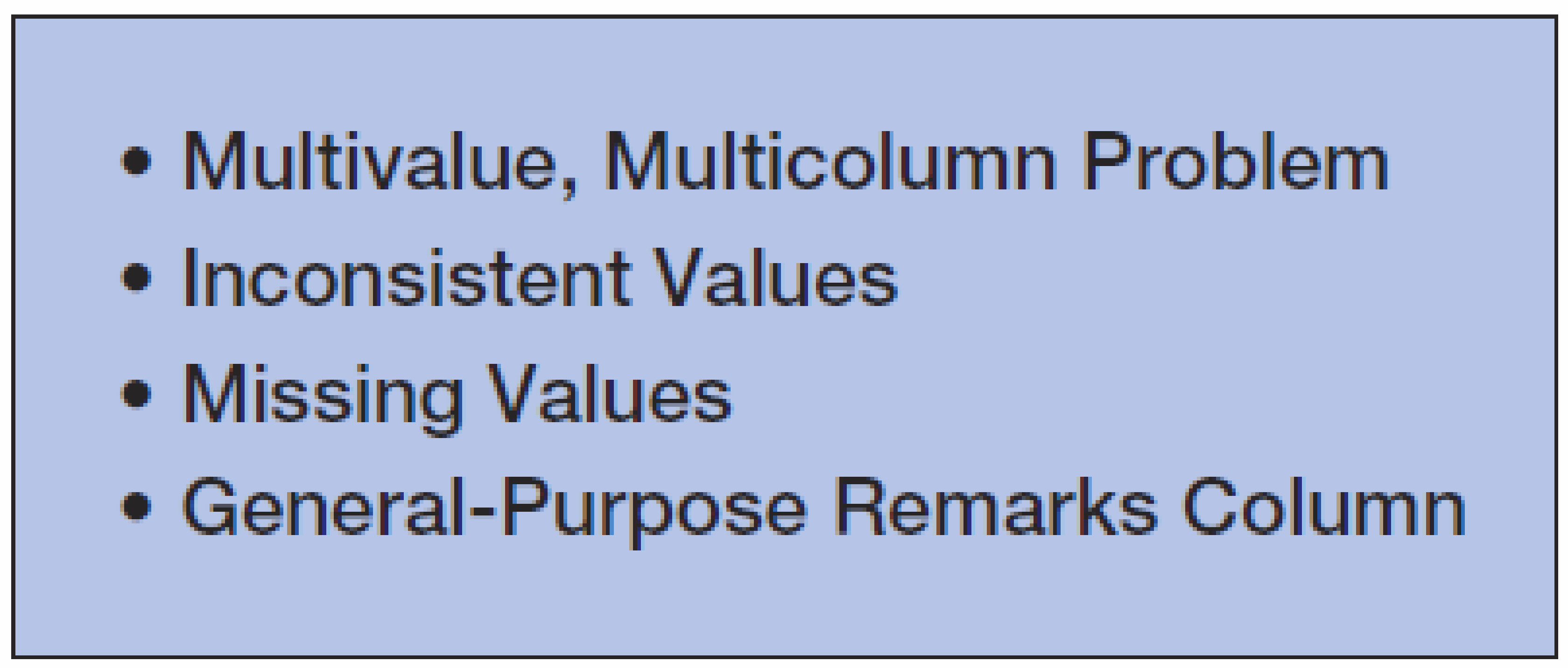
PRODUCT_PURCHASING (SKU, SKU_Description, VendorNumber, VendorName, VendorContact_1, VendorContact_2, VendorStreet, VendorCity, VendorState, VendorZip)

PRODUCT_USAGE (SKU, SKU_Description, QuantitySoldPastYear, QuantitySoldPastQuarter, QuantitySoldPastMonth)

PRODUCT_WEB (SKU, DetailPicture, ThumbnailPicture, MarketingShortDescription, MarketingLongDescription, PartColor)

PRODUCT_INVENTORY (SKU, PartNumber,

5. COMMON DESIGN PROBLEMS



- Multivalue, Multicolumn Problem
- Inconsistent Values
- Missing Values
- General-Purpose Remarks Column

The Multivalue, Multicolumn Problem

- The **multivalue, multicolumn problem** occurs when multiple values of an attribute are stored in more than one column:

  EMPLOYEE (EmployeeNumber, EmployeeLastName, EmployeeLastName, Email, Auto1_LicenseNumber, Auto2_LicenseNumber, Auto3_LicenseNumber)

- This is another form of a multivalued dependency.

- Solution = like the 4NF solution for multivalued dependencies, use a separate table to store the multiple values.

Inconsistent Values I

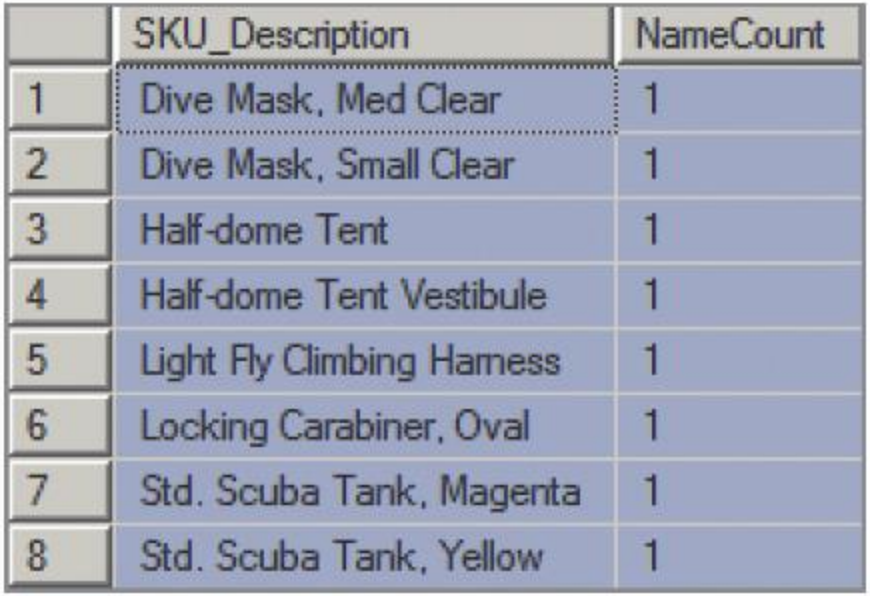
- **Inconsistent values** occur when different users, or different data sources, use slightly different forms of the same data value:
  - Different codings:
    - SKU_Description = 'Corn, Large Can'
    - SKU_Description = 'Can, Corn, Large'
    - SKU_Description = 'Large Can Corn‘
  - Different spellings:
    - Coffee, Cofee, Coffeee

Inconsistent Values II

- Particularly problematic are primary or foreign key values.
- To detect:
  - Use referential integrity check already discussed for checking keys.
  - Use the SQL GROUP BY clause on suspected columns.

Inconsistent Values III

```
SELECT     SKU_Description, COUNT(*) AS NameCount
FROM       SKU_DATA
GROUP BY   SKU_Description;
```



	SKU_Description	NameCount
1	Dive Mask, Med Clear	1
2	Dive Mask, Small Clear	1
3	Half-dome Tent	1
4	Half-dome Tent Vestibule	1
5	Light Fly Climbing Harness	1
6	Locking Carabiner, Oval	1
7	Std. Scuba Tank, Magenta	1
8	Std. Scuba Tank, Yellow	1

Missing Values

- A **missing value** or **null value** is a value that has never been provided.

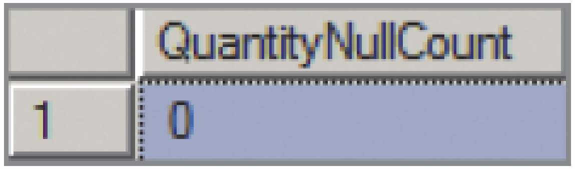
Null Values

- Null values are ambiguous:
  - May indicate that a value is inappropriate;
    - DateOfLastChildbirth is inappropriate for a male.
  - May indicate that a value is appropriate but unknown;
    - DateOfLastChildbirth is appropriate for a female, but may be unknown.
  - May indicate that a value is appropriate and known, but has never been entered;
    - DateOfLastChildbirth is appropriate for a female, and may be known but no one has recorded it in the database.

Checking for Null Values

- Use the SQL keyword IS NULL to check for null values:

```
SELECT     COUNT(*) AS QuantityNullCount
FROM       ORDER_ITEM
WHERE      Quantity IS NULL;
```



	QuantityNullCount
1	0

The General-Purpose Remarks Column

- A **general-purpose remarks column** is a column with a name such as:
  - Remarks
  - Comments
  - Notes
- It often contains important data stored in an inconsistent, verbal, and verbose way.
  - A typical use is to store data on a customer's interests.
- Such a column may:
  - Be used inconsistently
  - Hold multiple data items