6.2 Data Integrity Notes

Objective:

- Describe how data validation and data verification help protect integrity of data.
- Describe and use methods of data validation, Including range check, format check, length check, presence check, existence check, limit check and check digit.
- Describe and use methods of data verification during data entry and data transfer. During data entry including visual check, double entry. During data transfer including parity check (byte and block), checksum.

Data Integrity

- **Data Integrity** means **accuracy**, **completeness** and **consistency** of data.
- Data Integrity ensures that data received is same as data sent or data copied is same as the original.
- Data Integrity maintain validity of data, making sure data is correct and not corrupted.
- **Validation** and **verification** are two methods used to ensure **Data integrity**.

Accuracy (integrity) of data can be compromised by following reason;

- During **data entry** and **data transmission** stages.
- By **malicious attacks** on data, for example caused by malware and hacking
- By **accidental data loss** caused through hardware issues.

Validation

Validation is method of checking if entered data is **reasonable** and within a given **criteria** but it cannot check if data is **correct** or **accurate**. Validation is automatically carried out by computer software.

Example: if somebody accidentally enters their age as 62 instead of 26, it is reasonable but not accurate or correct.

Example of Validation Check:

- **Presence Check** to ensure that an entry field is not left blank.
- **Format Check**: It checks that characters entered conform to a pre-defined pattern. For example a date has to be dd/mm/yyyy
- **Length Check**: It checks that data contains an exact number of characters or numbers. For example with a telephone number.
- **Range Check**: It checks that only numbers within a **specified range** are accepted. For example month in a date must not exceed 12 .
- **Limit Check**: It check only one of the limit such as upper limit or lower limit. For example a maximum number of years for a person's age

- **Type check**: A type check will ensure that correct type of data is entered into that field. For example only a numeric value for the month in a date.
- **Existence Check**, for example that a file exists with filename referred to in data entry.

Verification

Verification is checking that data has **accurately** copied onto computer or **transferred** from one computer system to another. Verification can be used to make sure that data in your database is **correct.**

Verification is a way of **preventing errors** when data is entered manually (using keyboard) or when data is transferred from one computer to another.

Verification during Data Entry

When data is manually entered into a computer it needs to undergo verification to ensure there are **no errors**. There are **three ways** of doing this:

- **Double Entry:**

  Data is entered **twice**, computer compares both entries and outputs an error message requesting that data is entered again if they are different.

  Limitation: It is not ideal for **large amounts** of data. It could take a person a lot of time to enter **data twice**. They could enter **same mistake twice** and so it wouldn't get picked up.

- **Visual Check**:

  A SCREEN/VISUAL CHECK is a **manual check** completed by **user** who is entering data. When data entry is complete, data is displayed on screen and user is asked to confirm that it is correct before continuing.

- **Check Digits:**

  A check digit is **final digit** included in a code; it is calculated from all other digits in code.

  When a number is entered into computer (with check digit), the **check digit** is recalculated and, if same value (check digit) is not generated an error will occur.

  During check digit calculation, when a **single** check digit is allowed, letter **X** is used when remainder is calculated as 10.

  Note: When a series of **numbers** are used to identify something, it is possible to use a **check digit** method of **verification**.

  Application: Check digits are **used** in barcodes, product codes, International Standard Book Numbers (ISBN) and Vehicle Identification Numbers (VIN).

Types of error that can be detected by Check digit:

- **Phonetic Errors**, for example 13, thirteen, instead of 30, thirty.
- **Misplaced or Extra Digits**, for example 247 instead of 2407 or 42107 instead of 4207

- **Transposition Error** where two numbers have changed order, e.g 4087 instead of 4807.
- An **incorrect digit** entered, for example 5327 entered instead of 5307

Check Digit Calculation:

Now we will calculate check digit for 10 Digit ISBN **1 84146 201 ?**

Step # 1: Each digit in number is given a weighting of 10 -- 3, 2 or 1, starting from **left.**

ISBN	1	8	4	1	4	6	2	0	1
Code	10	9	8	7	6	5	4	3	2

Step #2 Digit is multiplied by its weighting and then each value is added to make a total.

ISBN	1	8	4	1	4	6	2	0	1
Code	10	9	8	7	6	5	4	3	2
Result	10	72	32	7	24	30	8	0	2

10 + 72 + 32 + 7 + 24 + 30 + 8 + 0 + 2 = 185

Step # 3 Total is divided by 11 and remainder subtracted from 11.

185 divide by 11 = 16 with 9 remaining.

Subtract remainder from 11.

11 - 9 = 2 Thus check digit is 2

Note: When this number is entered in other computer, check digit is recalculated and, if the same value is not generated, an error has occurred.

Verification during Data Transfer

When data is transferred electronically from one device to another, there is possibility of data corruption or even data loss. A number of ways exist to minimize this risk.

Checksums

Checksum is method to check if data has been changed or corrupted during data transmission. Data is sent in **blocks** and an **additional value**, checksum, is sent at end of block of data.

- If sum of other bytes in packet is **255 or less**, then checksum contains that exact value.
- If sum of other bytes is **more than 255**, then checksum is remainder of **total** value after it has been divided by 256.

Checksum Example:

Bytes total 1,151 1,151 / 256 = remainder 127 **Check Sum** = 127

At **sender end,** when block of data is about to be transmitted, checksum for bytes is first calculated. This value is transmitted with block of data. At **receiving end**, checksum is **recalculated** from block of data received. This calculated value is **compared** to checksum transmitted. If they are **same**, then data was transmitted without any errors; if they are different, then a request is sent for data to be re-transmitted.

Parity checks

Parity check is method to check whether data has been changed or corrupted following transmission from one device or medium to another.

Example:

A byte of data is allocated a **parity bit**. This is allocated before transmission. Systems that use **even parity** have an even number of 1-bits; systems that use **odd parity** have an odd number of 1-bits.

	1	1	0	1	1	0	0

If this byte is using **even parity**, then parity bit needs to be 0. If **odd parity** is being used, then parity bit needs to be 1.

Therefore, byte just **before transmission** would be

either (even parity):	0	1	1	0	1	1	0	0
	parity bit							
or (odd parity):	1	1	1	0	1	1	0	0

Before data is transferred, agreement is made between sender and receiver regarding which of the two types of parity are used. If a byte has been transmitted, and **even** parity is used, an **error** would be **flagged** if byte now had an odd number of 1-bits at **receiver's end**. This means an error has occurred during transmission of data.

Limitation of Parity Bit:

If there are multiple errors in same byte/column, that still produce same parity bit, error will not be detected.

Example: If by chance, two errors occur in byte being transmitted "00001111" changes to "00000011" this error won't be detected by even parity.

Parity Block checks

At **Sender End**, program reads a group of seven bytes. Data is represented by **seven bits** for each byte. Most significant bit in each byte, bit 7, is undefined so we have left it blank.

	Seven-bit codes						
	1	0	1	0	0	1	1
	0	1	1	0	0	0	1
	1	0	1	1	0	0	0
	0	0	1	1	1	0	0
	0	1	1	0	0	1	0
	0	1	1	0	0	0	1
	0	1	1	0	0	0	1

Parity bit is set for each of bytes. Most significant bit is set to achieve **even parity**.

Parity bits	Seven-bit codes						
0	1	0	1	0	0	1	1
1	0	1	1	0	0	0	1
1	1	0	1	1	0	0	0
1	0	0	1	1	1	0	0
1	0	1	1	0	0	1	0
1	0	1	1	0	0	0	1
1	0	1	1	0	0	0	1

Additional byte is then created and each bit is set as parity bit for the bits at that bit position. This includes counting the parity bits in the seven bytes containing data.

Parity bits	Seven-bit codes						
0	1	0	1	0	0	1	1
1	0	1	1	0	0	0	1
1	1	0	1	1	0	0	0
1	0	0	1	1	1	0	0
1	0	1	1	0	0	1	0
1	0	1	1	0	0	0	1
1	0	1	1	0	0	0	1
0	0	0	1	0	1	0	0

← Parity byte

Program then transmits the eight bytes in sequence.

At **receiving end**, a program takes the **eight bytes** as input and checks the parity sums for the individual bytes and for the bit positions.

If there is just one error in seven bytes this method will allow program at receiving end to identify position of error. It can **correct** the error so the transmission can be accepted.

Difference between data security and data integrity

- **Integrity** deals with validity of data, data should be free from errors while **Security** deals with protection of data.
- **Security** protects data from illegal access/loss while **Integrity** deals with making sure data is not corrupted after input or transmission.

****