Learnings from project BanckingSyste

① Char(n) : Fixed length, always takes 'n' bytes
VarChar(n) : Variable length, saves space.

② We can create a secure random number ~~&s~~ of length 19 & can save them in Set to avoid duplication.

≡ Code ≡

```
SecureRandom random = new SecureRandom();
// random number generator for better randomness.
Set <long> generatedNumbers = new HashSet<>();
// stores already generatedNumbers to ensure no duplicates
long number;
do {
number = 1000000000000000000L + (MATHS.abs(random.nextLong()) % 9000000000000000000L);
                                                                         }
// random.nextLong() -> Generates a random long number.
// %9---- L -> ensures within a 19-digit range.
// Adds 10---- L -> ensures it's always 19 digits long.
while(!generatedNumber.add(number));
 // if nb is not in Set loop exits & number added.
 //    ,,    is in Set loop runs again
```

③ .equals is applied when it is String probably but it is not applied on integer.

④ You declared getNumber but did not ~~realize~~ initialize it, causing an error. You must assign a value before returning it:

```
                    public long num() {.
long ~~el~~ getNumber; ✗
   return getNumber; }
```

* If nextLine(); in code doesnt work (doesn't take inputs). than it is better to write an extra: to clear the buffer before calling nextLine();
 * Use sk. next line (); immediatly after any nextInt nextDouble(), or smilar methods to clear buffer.

* Printf() formatting

%→ Used as placeholder in printf() to format output.

%d → for integers (eg. reservation ID, room number).

%S → for Strings (eg guestime, contact Numbers).

-14d → -14 means left-aligned & 14 spaces wide for inte

-15S → -15 " " " " " String

Code

```
S.O.P("|%-10d |%-15s|%-10d|\n", 1, "John", 10
```

Output

```
| 1          | John| 101.
```

* .getTimestamp () used in JDBC to retrieve timestamp values (date and time) from a DB result set.

Code

```
String reservationDate = rs.getTimestamp("rereservation_date).toString
```

* .next() moves cursor to next ~~line~~ row in a Result Set and returns "true" if a row exists, otherwise returns "false"

→ Your query must include accountNumber in SELECT statement, otherwise, you can't fetch it.

Means

SELECT email_id, pin, accountNumber FROM account

→ Setting connection.setAutoCommit(false); affects the entire connection object, not just function. It stays false until explicitly set to true.

~~It is go to use~~ → It is a good practice; to set autoCommit(true) in finally to restore default behaviour & avoid unexpected transaction handling.

→ (Update query) needs connection.commit(); b/c it modifies database.

(SELECT query) doesn't Not need connection.commit(); b/c it only reads data.

→ If an exception occurs before closing PreparedStatement it stays open, causing memory leaks.

So it is good to use try-with-resource so Java automatically closes PreparedStatement after use.

Note:- Catch is for handling exceptions explicitly, catch is not required in try-with-resources.

{ Generally, TWR for resources that needs to be closed eg, Connection, ResultSet etc.

→ Problem:- Scanner.nextLong() leaves a new line in buffer, causing issues with subsequent nextLine();

Solution: Add scanner.nextLine(); right after scanner.nextLong to consume the leftover newline.

try{ connection.set~~Autocommit(false~~);

→ Issue : If an exception occurs before commit(), changes are not rolled back.
Fix:- Ensure rollback(); is called inside catch.
→ TWR General Rule. (Try-with Resources.
① → TWR needs a catch when its not inside another try - catch. b/c it only closes resources but doesn't handle exception.
② TWR is more effecient than manually calling scanner.close(); b/c it ensures proper resource mgt even if an exception occurs.
→ Scanner.nextLong(); crashes if the user enters non-numeric input.
Use a loop & in which user enters a valid (non- nongative prompt) by repeatedly prompting until valid long is provided.
→ Scanner.next in this code consumes the invalid input.