Batch: b2	**Roll No.:** 110 and 109
Experiment / assignment / tutorial No. 5	

Title: Implementing TCL/DCL

Objective: To be able to Implement TCL and DCL.

Expected Outcome of Experiment:
CO 2: Convert entity-relationship diagrams into relational tables, populate a relational database and formulate SQL queries on the data Use SQL for creation and query the database.
CO 4: Demonstrate the concept of transaction, concurrency control and recovery techniques.

Books/ Journals/ Websites referred:

1. Dr. P.S. Deshpande, SQL and PL/SQL for Oracle 10g.Black book, Dreamtech Press
2. www.db-book.com
3. Korth, Slberchatz, Sudarshan : “Database Systems Concept”, 5th Edition , McGraw Hill
4. Elmasri and Navathe,”Fundamentals of database Systems”, 4th Edition,PEARSON Education.

Resources used: PostgreSQL

Theory

DCL stands for Data Control Language.

DCL is used to control user access in a database.

This command is related to the security issues.

Using DCL command, it allows or restricts the user from accessing data in database schema.

DCL commands are as follows,

GRANT

REVOKE

It is used to grant or revoke access permissions from any database user.

GRANT command gives user's access privileges to the database.

This command allows specified users to perform specific tasks.

Syntax:

```
GRANT { { SELECT | INSERT | UPDATE | DELETE | TRUNCATE |
REFERENCES | TRIGGER }
    [, ...] | ALL [ PRIVILEGES ] }
    ON { [ TABLE ] table_name [, ...]
         | ALL TABLES IN SCHEMA schema_name [, ...] }
    TO { [ GROUP ] role_name | PUBLIC } [, ...] [ WITH GRANT
OPTION ]

GRANT { { SELECT | INSERT | UPDATE | REFERENCES } ( column_name
[, ...] )
    [, ...] | ALL [ PRIVILEGES ] ( column_name [, ...] ) }
    ON [ TABLE ] table_name [, ...]
    TO { [ GROUP ] role_name | PUBLIC } [, ...] [ WITH GRANT
OPTION ]
```

Example

```
GRANT INSERT ON films TO PUBLIC;
GRANT ALL PRIVILEGES ON kinds TO ram;
GRANT admins TO krishna;
```

REVOKE command is used to cancel previously granted or denied permissions.

This command withdraw access privileges given with the GRANT command.

It takes back permissions from user.

Syntax:

```
REVOKE [ GRANT OPTION FOR ]
    { { SELECT | INSERT | UPDATE | DELETE | TRUNCATE |
REFERENCES | TRIGGER }
    [, ...] | ALL [ PRIVILEGES ] }
    ON { [ TABLE ] table_name [, ...]
         | ALL TABLES IN SCHEMA schema_name [, ...] }
    FROM { [ GROUP ] role_name | PUBLIC } [, ...]
    [ CASCADE | RESTRICT ]
```

```
REVOKE [ GRANT OPTION FOR ]
    { { SELECT | INSERT | UPDATE | REFERENCES } ( column_name
[, ...] )
    [, ...] | ALL [ PRIVILEGES ] ( column_name [, ...] ) }
    ON [ TABLE ] table_name [, ...]
    FROM { [ GROUP ] role_name | PUBLIC } [, ...]
    [ CASCADE | RESTRICT ]

REVOKE [ GRANT OPTION FOR ]
    { { USAGE | SELECT | UPDATE }
    [, ...] | ALL [ PRIVILEGES ] }
    ON { SEQUENCE sequence_name [, ...]
         | ALL SEQUENCES IN SCHEMA schema_name [, ...] }
    FROM { [ GROUP ] role_name | PUBLIC } [, ...]
    [ CASCADE | RESTRICT ]
```

Example

```
REVOKE INSERT ON films FROM PUBLIC;
REVOKE ALL PRIVILEGES ON kinds FROM Madhav;
REVOKE admins FROM Keshav;
```

TCL stands for **Transaction Control Language.**

This command is used to manage the changes made by DML statements.

TCL allows the statements to be grouped together into logical transactions.

TCL commands are as follows:
1. COMMIT
2. SAVEPOINT
3. ROLLBACK
4. SET TRANSACTION

COMMIT command saves all the work done. It ends the current transaction and makes permanent changes during the transaction

Syntax:
commit;

SAVEPOINT command is used for saving all the current point in the processing of a transaction. It marks and saves the current point in the processing of a transaction. It is used to temporarily save a transaction, so that you can rollback to that point whenever necessary.

Syntax

```
SAVEPOINT savepoint_name
```

ROLLBACK command restores database to original since the last COMMIT. It is used to restores the database to last committed state.

Syntax:

```
ROLLBACK [ WORK | TRANSACTION ] TO [ SAVEPOINT ]
savepoint_name
```

Example

```
BEGIN;
    INSERT INTO table1 VALUES (1);
    SAVEPOINT my_savepoint;
    INSERT INTO table1 VALUES (2);
    ROLLBACK TO SAVEPOINT my_savepoint;
    INSERT INTO table1 VALUES (3);
COMMIT;
```

The above transaction will insert the values 1 and 3, but not 2.

SET TRANSACTION is used for placing a name on a transaction. You can specify a transaction to be read only or read write. This command is used to initiate a database transaction.

Syntax:

SET TRANSACTION [Read Write | Read Only];

The SET TRANSACTION command sets the characteristics of the current transaction. It has no effect on any subsequent transactions. SET SESSION CHARACTERISTICS sets the default transaction characteristics for subsequent transactions of a session. These defaults can be overridden by SET TRANSACTION for an individual transaction.

The available transaction characteristics are the transaction isolation level, the transaction access mode (read/write or read-only), and the deferrable mode. In addition, a snapshot can be selected, though only for the current transaction, not as a session default.

The isolation level of a transaction determines what data the transaction can see when other transactions are running concurrently:

READ COMMITTED

A statement can only see rows committed before it began. This is the default.

REPEATABLE READ

All statements of the current transaction can only see rows committed before the first query or data-modification statement was executed in this transaction.

SERIALIZABLE

All statements of the current transaction can only see rows committed before the first query or data-modification statement was executed in this transaction. If a pattern of reads and writes among concurrent serializable transactions would create a situation which could not have occurred for any serial (one-at-a-time) execution of those transactions, one of them will be rolled back with a serialization_failure error.

Examples

With the default read committed isolation level.

```
process A: BEGIN; -- the default is READ COMMITED

process A: SELECT sum(value) FROM purchases;
--- process A sees that the sum is 1600

process B: INSERT INTO purchases (value) VALUES (400)
--- process B inserts a new row into the table while
--- process A's transaction is in progress

process A: SELECT sum(value) FROM purchases;
--- process A sees that the sum is 2000

process A: COMMIT;
```

If we want to avoid the changing sum value in process A during the lifespan of the transaction, we can use the repeatable read transaction mode.

```
process A: BEGIN TRANSACTION ISOLATION LEVEL REPEATABLE READ;
process A: SELECT sum(value) FROM purchases;
--- process A sees that the sum is 1600

process B: INSERT INTO purchases (value) VALUES (400)
--- process B inserts a new row into the table while
--- process A's transaction is in progress

process A: SELECT sum(value) FROM purchases;
--- process A still sees that the sum is 1600

process A: COMMIT;
```

The transaction in process A fill freeze its snapshot of the data and offer consistent values during the life of the transaction.

Repeatable reads are not more expensive than the default read commit transaction. There is no need to worry about performance penalties. However, applications must be prepared to retry transactions due to serialization failures.

Let's observe an issue that can occur while using the repeatable read isolation level — the `could not serialize access due to concurrent update` error.

```
process A: BEGIN TRANSACTION ISOLATION LEVEL REPEATABLE READ;
process B: BEGIN;
process B: UPDATE purchases SET value = 500 WHERE id = 1;
process A: UPDATE purchases SET value = 600 WHERE id = 1;
-- process A wants to update the value while process B is changing it
-- process A is blocked until process B commits

process B: COMMIT;
process A: ERROR:  could not serialize access due to concurrent update
-- process A immidiatly errors out when process B commits
```

If process B would rolls back, then its changes are negated and repeatable read can proceed without issues. However, if process B commits the changes then the repeatable read transaction will be rolled back with the error message because it can not modify or lock the rows changed by other processes after the repeatable read transaction has began.

demonstrate the differences between the two isolation modes.

```
process A: BEGIN TRANSACTION ISOLATION LEVEL REPEATABLE READ;
process A: SELECT sum(value) FROM purchases;
process A: INSERT INTO purchases (value) VALUES (100);
process B: BEGIN TRANSACTION ISOLATION LEVEL REPEATABLE READ;
process B: SELECT sum(value) FROM purchases;
process B: INSERT INTO purchases (id, value);
process B: COMMIT;
process A: COMMIT;
```

With Repeatable Reads everything works, but if we run the same thing with a Serializable isolation mode, process A will error out.

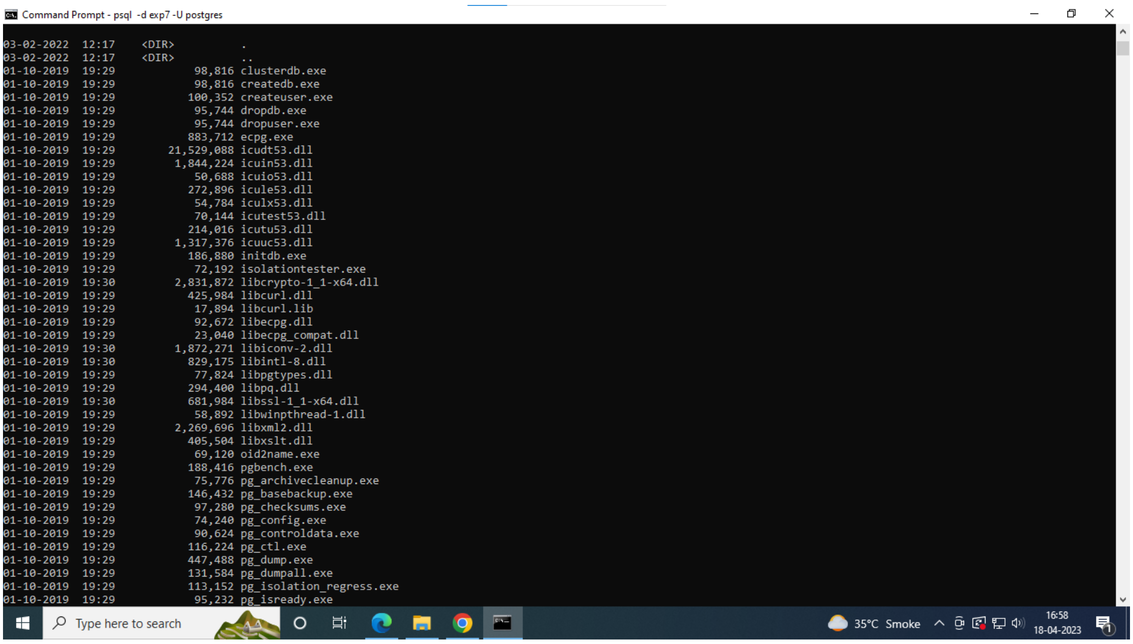
```
process A: BEGIN TRANSACTION ISOLATION LEVEL SERIALIZABLE;
process A: SELECT sum(value) FROM purchases;
process A: INSERT INTO purchases (value) VALUES (100);
process B: BEGIN TRANSACTION ISOLATION LEVEL SERIALIZABLE;
process B: SELECT sum(value) FROM purchases;
process B: INSERT INTO purchases (id, value);
process B: COMMIT;
process A: COMMIT;
ERROR: could not serialize access due to read/write
dependencies among transactions
```

```
DETAIL: Reason code: Canceled on identification as
a pivot, during commit attempt.
HINT: The transaction might succeed if retried.
```

Both transactions have modified what the other transaction would have read in the select statements. If both would allow to commit this would violate the Serializable behaviour, because if they were run one at a time, one of the transactions would have seen the new record inserted by the other transaction.

Implementation Screenshots (Problem Statement, Query and Screenshots of Results):

Demonstrate DCL and TCL language commands on your database.



```
Select Command Prompt - psql  -d exp7 -U postgres
01-10-2019  19:29           102,400 pg_resetwal.exe
01-10-2019  19:29           209,408 pg_restore.exe
01-10-2019  19:29           135,168 pg_rewind.exe
01-10-2019  19:29            71,680 pg_standby.exe
01-10-2019  19:29            80,896 pg_test_fsync.exe
01-10-2019  19:29            71,680 pg_test_timing.exe
01-10-2019  19:29           173,568 pg_upgrade.exe
01-10-2019  19:29           130,048 pg_waldump.exe
01-10-2019  19:29         7,418,368 postgres.exe
01-10-2019  19:29           542,208 psql.exe
01-10-2019  19:29           102,400 reindexdb.exe
01-10-2019  19:29           351,816 stackbuilder.exe
01-10-2019  19:29           107,008 vacuumdb.exe
01-10-2019  19:29            68,608 vacuumlo.exe
01-10-2019  19:29           147,456 wxbase28u_net_vc_custom.dll
01-10-2019  19:29         1,377,280 wxbase28u_vc_custom.dll
01-10-2019  19:29           148,480 wxbase28u_xml_vc_custom.dll
01-10-2019  19:29           846,336 wxmsw28u_adv_vc_custom.dll
01-10-2019  19:29           382,976 wxmsw28u_aui_vc_custom.dll
01-10-2019  19:29         3,443,200 wxmsw28u_core_vc_custom.dll
01-10-2019  19:29           569,344 wxmsw28u_html_vc_custom.dll
01-10-2019  19:29           620,032 wxmsw28u_xrc_vc_custom.dll
01-10-2019  19:29            96,256 zic.exe
01-10-2019  19:29            86,528 zlib1.dll
              69 File(s)     56,250,048 bytes
               2 Dir(s)  52,556,668,928 bytes free

C:\Program Files\PostgreSQL\12\bin>psql -d exp7 -U postgres
Password for user postgres:
psql: error: could not connect to server: FATAL:  password authentication failed for user "postgres"

C:\Program Files\PostgreSQL\12\bin>psql -d exp7 -U postgres
Password for user postgres:
psql (12.0)
WARNING: Console code page (437) differs from Windows code page (1252)
         8-bit characters might not work correctly. See psql reference
         page "Notes for Windows users" for details.
Type "help" for help.

exp7=# \dt
Did not find any relations.
exp7=#
Type here to search    35°C Smoke    16:59 18-04-2023
```

```
C:\Program Files\PostgreSQL\12\bin>psql -d KJSCE_CFAS -U postgres
Password for user postgres:
psql (12.0)
WARNING: Console code page (437) differs from Windows code page (1252)
         8-bit characters might not work correctly. See psql reference
         page "Notes for Windows users" for details.
Type "help" for help.

KJSCE_CFAS=# \dt
              List of relations
 Schema |      Name       | Type  |  Owner
--------+-----------------+-------+----------
 public | course_feedback | table | postgres
 public | faculty         | table | postgres
 public | student         | table | postgres
(3 rows)


KJSCE_CFAS=# _
```

```
Command Prompt - psql  -d KJSCE_CFAS -U postgres
Password for user Student:
psql: error: could not connect to server: FATAL:  password authentication failed for user "Student"

C:\Program Files\PostgreSQL\12\bin>psql -d KJSCE_CFAS _U postgres
psql: warning: extra command-line argument "postgres" ignored
Password for user _U:
psql: error: could not connect to server: FATAL:  password authentication failed for user "_U"

C:\Program Files\PostgreSQL\12\bin>psql -d KJSCE_CFAS -U postgres
Password for user postgres:
psql (12.0)
WARNING: Console code page (437) differs from Windows code page (1252)
         8-bit characters might not work correctly. See psql reference
         page "Notes for Windows users" for details.
Type "help" for help.

KJSCE_CFAS=# help
You are using psql, the command-line interface to PostgreSQL.
Type:  \copyright for distribution terms
       \h for help with SQL commands
       \? for help with psql commands
       \g or terminate with semicolon to execute query
       \q to quit
KJSCE_CFAS=# create user pikachu with password 'sushant';
CREATE ROLE
KJSCE_CFAS=# \du
                                   List of roles
 Role name |                         Attributes                         | Member of
-----------+------------------------------------------------------------+-----------
 pikachu   |                                                            | {}
 postgres  | Superuser, Create role, Create DB, Replication, Bypass RLS | {}


KJSCE_CFAS=# grant all on student to pikachu;
GRANT
KJSCE_CFAS=# \du
                                   List of roles
 Role name |                         Attributes                         | Member of
-----------+------------------------------------------------------------+-----------
 pikachu   |                                                            | {}
 postgres  | Superuser, Create role, Create DB, Replication, Bypass RLS | {}


KJSCE_CFAS=# _
Type here to search    Watchlist ideas    17:13 18-04-2023
```

```
Microsoft Windows [Version 10.0.19044.2130]
(c) Microsoft Corporation. All rights reserved.

C:\Users\Student>cd\prog*

C:\Program Files>cd post*

C:\Program Files\PostgreSQL>cd 12

C:\Program Files\PostgreSQL\12>cd bin

C:\Program Files\PostgreSQL\12\bin>psql -d KJSCE_CFAS -U pikachu
Password for user pikachu:
psql (12.0)
WARNING: Console code page (437) differs from Windows code page (1252)
         8-bit characters might not work correctly. See psql reference
         page "Notes for Windows users" for details.
Type "help" for help.

KJSCE_CFAS=> select * from student
KJSCE_CFAS-> ;
 roll_no |        email        |   pass    | fname | lname | gender |    dob     | adm_yr | yr | sem | course_id | branch | fsn
---------+---------------------+-----------+-------+-------+--------+------------+--------+----+-----+-----------+--------+-----
      96 | jkl.pqr@somaiya.edu | jP123456  | JKL   | PQR   | F      | 2004-10-01 |   2022 |  1 |   2 |           |        |
      16 | abc.ghi@somaiya.edu | bB123456  | ABC   | GHI   | M      | 2003-09-01 |   2021 |  2 |   4 |           | COMP   |
       2 | stu.vwx@somaiya.edu | sV123456  | STU   | VWX   | F      | 2002-10-01 |   2020 |  3 |   6 |           |        |
       3 | yza.bcd@somaiya.edu | yB123456  | YZA   | BCD   | F      | 2002-09-01 |   2020 |  3 |   6 |           |        |
       4 | efg.hij@somaiya.edu | eH123456  | EFG   | HIJ   | M      | 2002-11-01 |   2020 |  3 |   6 |           |        |
       5 | klm.nop@somaiya.edu | kN123456  | KLM   | NOP   | M      | 2002-10-01 |   2020 |  3 |   6 |           |        |
(6 rows)


KJSCE_CFAS=> ;
KJSCE_CFAS=> select * from student;
ERROR:  permission denied for table student
KJSCE_CFAS=> select * from student;
 roll_no |        email        |   pass    | fname | lname | gender |    dob     | adm_yr | yr | sem | course_id | branch | fsn
---------+---------------------+-----------+-------+-------+--------+------------+--------+----+-----+-----------+--------+-----
      96 | jkl.pqr@somaiya.edu | jP123456  | JKL   | PQR   | F      | 2004-10-01 |   2022 |  1 |   2 |           |        |
      16 | abc.ghi@somaiya.edu | bB123456  | ABC   | GHI   | M      | 2003-09-01 |   2021 |  2 |   4 |           | COMP   |
       2 | stu.vwx@somaiya.edu | sV123456  | STU   | VWX   | F      | 2002-10-01 |   2020 |  3 |   6 |           |        |
       3 | yza.bcd@somaiya.edu | yB123456  | YZA   | BCD   | F      | 2002-09-01 |   2020 |  3 |   6 |           |        |
       4 | efg.hij@somaiya.edu | eH123456  | EFG   | HIJ   | M      | 2002-11-01 |   2020 |  3 |   6 |           |        |
       5 | klm.nop@somaiya.edu | kN123456  | KLM   | NOP   | M      | 2002-10-01 |   2020 |  3 |   6 |           |        |
```

```
Type "help" for help.

KJSCE_CFAS=> grant all from student to pikachu;
ERROR:  syntax error at or near "from"
LINE 1: grant all from student to pikachu;
                  ^
KJSCE_CFAS=> grant all on student to pikachu;
WARNING:  no privileges were granted for "student"
GRANT
KJSCE_CFAS=> \du
                                   List of roles
 Role name |                         Attributes                         | Member of
-----------+------------------------------------------------------------+-----------
 pikachu   |                                                            | {}
 postgres  | Superuser, Create role, Create DB, Replication, Bypass RLS | {}


KJSCE_CFAS=> exit

C:\Program Files\PostgreSQL\12\bin>psql -d KJSCE_CFAS -U postgres
Password for user postgres:
psql (12.0)
WARNING: Console code page (437) differs from Windows code page (1252)
         8-bit characters might not work correctly. See psql reference
         page "Notes for Windows users" for details.
Type "help" for help.

KJSCE_CFAS=# grant all on student to pikachu;
GRANT
KJSCE_CFAS=# \du
                                   List of roles
 Role name |                         Attributes                         | Member of
-----------+------------------------------------------------------------+-----------
 pikachu   |                                                            | {}
 postgres  | Superuser, Create role, Create DB, Replication, Bypass RLS | {}


KJSCE_CFAS=# revoke all on student from pikachu;
REVOKE
KJSCE_CFAS=# grant all on student to pikachu;
GRANT
KJSCE_CFAS=#
```

```
KJSCE_CFAS=# grant select on student to pikachu
KJSCE_CFAS-# ;
GRANT
KJSCE_CFAS=# revoke select on student from pikachu;
REVOKE
KJSCE_CFAS=#
```

```
KJSCE_CFAS=> DELETE FROM student where sem =2;
DELETE 1
KJSCE_CFAS=> DELETE FROM student where sem =2;
ERROR:  permission denied for table student
KJSCE_CFAS=>
```

```
KJSCE_CFAS=> select * from student;
 roll_no |        email         |   pass    | fname | lname | gender |    dob     | adm_yr | yr | sem | course_id | branch | fsn
---------+----------------------+-----------+-------+-------+--------+------------+--------+----+-----+-----------+--------+-----
      96 | jkl.pqr@somaiya.edu | jP123456 | JKL   | PQR   | F      | 2004-10-01 |   2022 |  1 |   2 |           |        |
      16 | abc.ghi@somaiya.edu | bB123456 | ABC   | GHI   | M      | 2003-09-01 |   2021 |  2 |   4 |           | COMP   |
       2 | stu.vwx@somaiya.edu | sV123456 | STU   | VWX   | F      | 2002-10-01 |   2020 |  3 |   6 |           |        |
       3 | yza.bcd@somaiya.edu | yB123456 | YZA   | BCD   | F      | 2002-09-01 |   2020 |  3 |   6 |           |        |
       4 | efg.hij@somaiya.edu | eH123456 | EFG   | HIJ   | M      | 2002-11-01 |   2020 |  3 |   6 |           |        |
       5 | klm.nop@somaiya.edu | kN123456 | KLM   | NOP   | M      | 2002-10-01 |   2020 |  3 |   6 |           |        |
(6 rows)

KJSCE_CFAS=> select * from student;
ERROR:  permission denied for table student
KJSCE_CFAS=>
```

```
KJSCE_CFAS=# grant DELETE on student to pikachu;
GRANT
KJSCE_CFAS=# grant select on student to pikachu;
GRANT
KJSCE_CFAS=# revoke all on student from pikachu;
REVOKE
KJSCE_CFAS=#
```

Conclusion:

Postlab question:

1. Discuss ACID properties of transaction with suitable example

ACID stands for Atomicity, Consistency, Isolation, and Durability, which are the four properties that ensure reliability and consistency in database transactions.

Atomicity: This property ensures that a transaction is treated as a single unit of work, which either completes in its entirety or fails entirely. This means that if any part of the transaction fails, the entire transaction is rolled back to its initial state, leaving the database unchanged.

For example, suppose you want to transfer funds from one bank account to another. The transaction must be atomic to ensure that the money is either transferred successfully or not transferred at all. If the transaction fails in the middle, the funds should be returned to the original account, and the second account should not receive any money.

Consistency: This property ensures that a transaction brings the database from one valid state to another valid state. The database must be consistent before and after a transaction.

For example, suppose you have a database of student records. If a transaction changes a student's name from "aatmaj" to "rohan," the database must remain consistent. This means that all other records and transactions that refer to "aatmaj" should also be updated to "rohan."

Isolation: This property ensures that concurrent transactions do not interfere with each other. Transactions should be executed in isolation, and the final outcome should be the same as if they were executed sequentially.

For example, suppose two transactions are concurrently updating the same bank account. Isolation ensures that both transactions will be executed as if they were executed sequentially, and the final balance in the account will be correct.

Durability: This property ensures that once a transaction is committed, it is permanently stored in the database and can survive subsequent failures, such as power outages or system crashes.

For example, suppose you have completed a transaction to update a database record. Durability ensures that the updated record is permanently saved and can be retrieved even if the system crashes immediately after the transaction was committed.

Overall, the ACID properties ensure that database transactions are reliable, consistent, and safe from data corruption or loss.