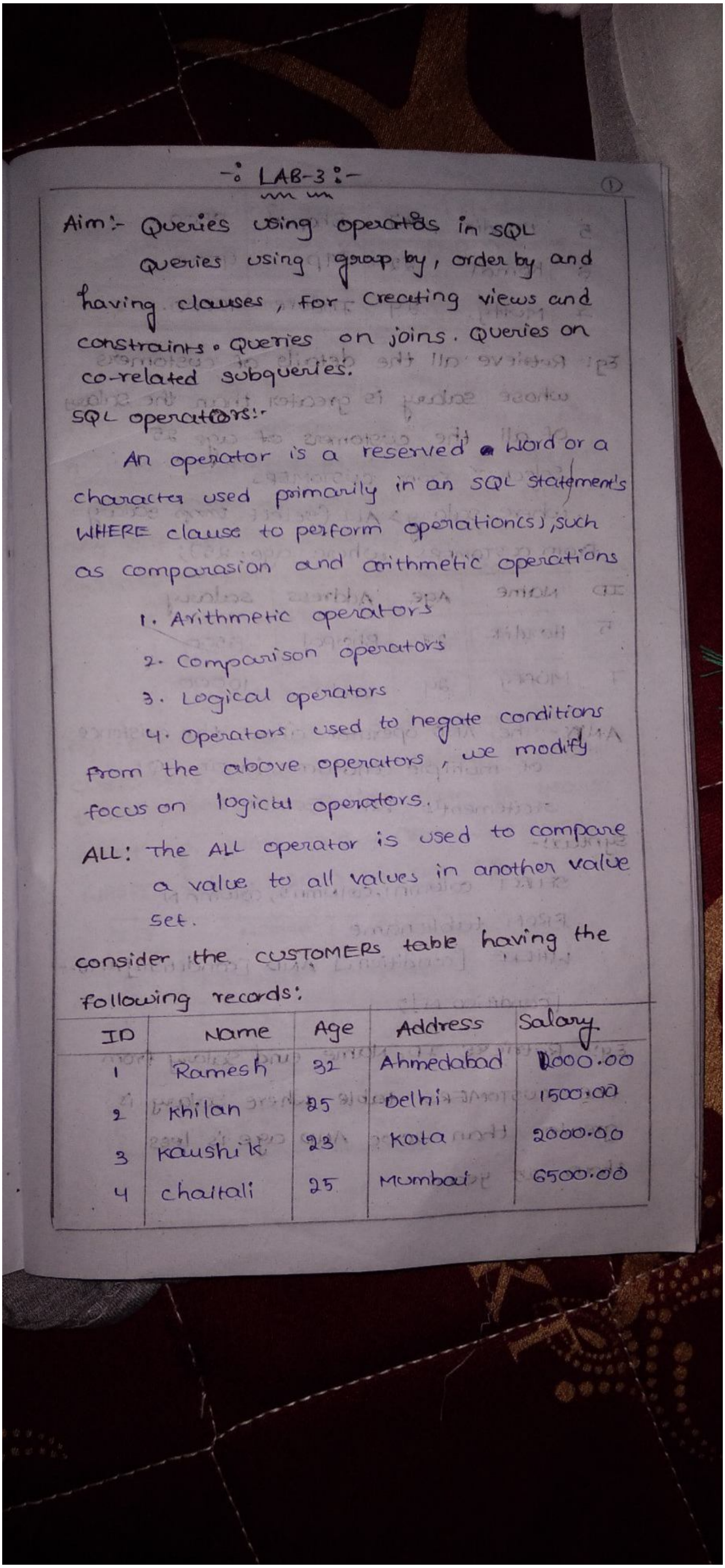


-: LAB-3 :-

Aim:- Queries using operators in SQL

Queries using group by, order by and having clauses, for creating views and constraints. Queries on joins. Queries on co-related subqueries.

SQL operators:-

An operator is a reserved word or a character used primarily in an SQL statement's WHERE clause to perform operation(s), such as comparasion and arithmetic operations

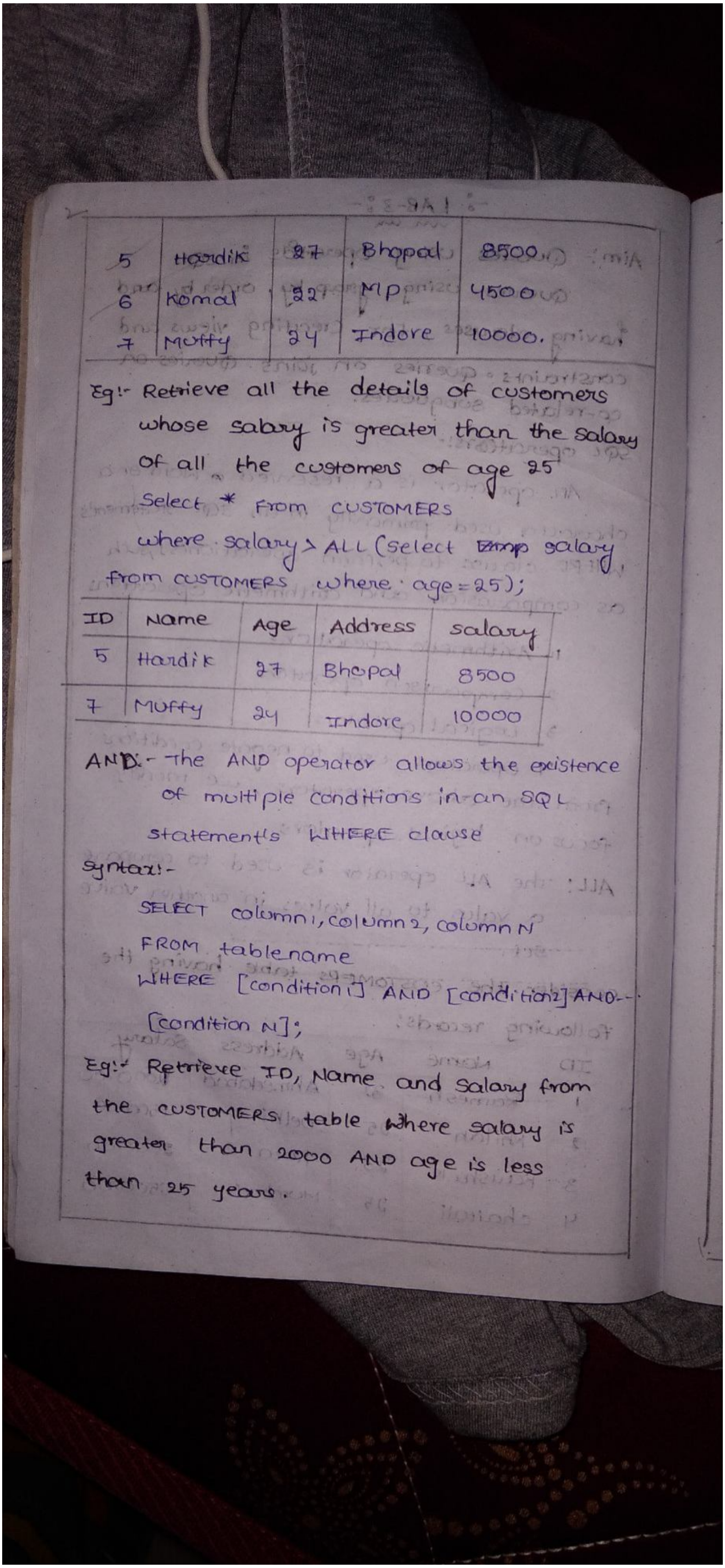
1. Arithmetic operators
2. Comparison operators
3. Logical operators
4. Operators used to negate conditions

From the above operators, we modify focus on logical operators.

ALL: The ALL operator is used to compare a value to all values in another value set.

consider the CUSTOMERS table having the following records:

ID	Name	Age	Address	Salary
1	Ramesh	32	Ahmedabad	2000.00
2	khilan	25	Delhi	1500.00
3	kaushik	23	kota	2000.00
4	chaitali	25	Mumbai	6500.00



5	Hardik	27	Bhopal	8500
6	Komal	22	MP	4500
7	Muffy	24	Indore	10000.

Eg:- Retrieve all the details of customers whose salary is greater than the salary of all the customers of age 25

```
Select * From CUSTOMERS
where salary > ALL (Select ~~Emp~~ salary
from CUSTOMERS where age=25);
```

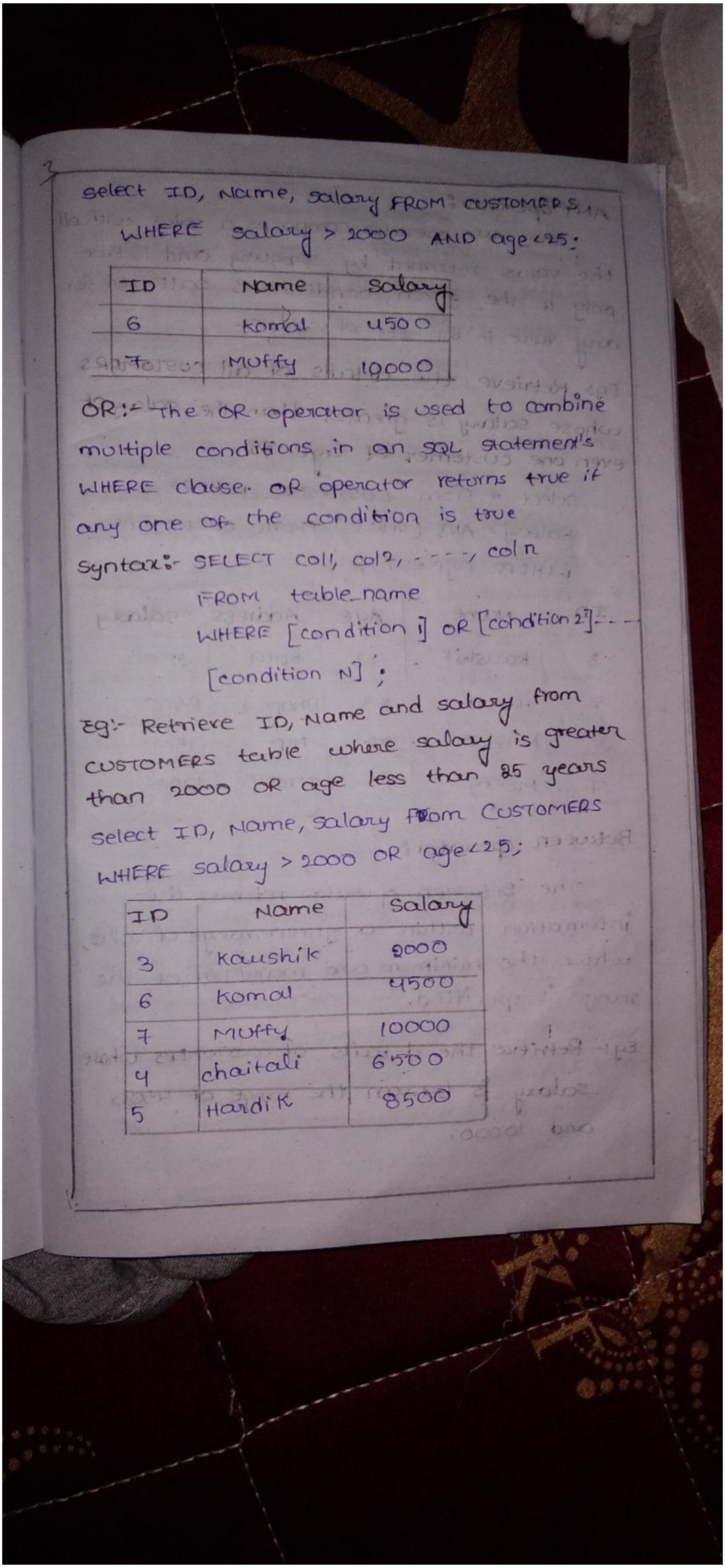
ID	Name	Age	Address	salary
5	Hardik	27	Bhopal	8500
7	Muffy	24	Indore	10000

AND:- The AND operator allows the existence of multiple conditions in an SQL statement's WHERE clause

syntax:-

```
SELECT column1, column2, column N
FROM tablename
WHERE [condition1] AND [condition2] AND--
[condition N];
```

Eg:- Retrieve ID, Name and salary from the CUSTOMERS table where salary is greater than 2000 AND age is less than 25 years.



```
select ID, Name, salary FROM CUSTOMERS
  WHERE salary > 2000 AND age<25;
```

ID	Name	salary
6	Komal	4500
7	Muffy	10000

OR:- The OR operator is used to combine multiple conditions in an SQL statement's WHERE clause. OR operator returns true if any one of the condition is true

```
Syntax:- SELECT col1, col2, ----, col n
            FROM teble_name
            WHERE [condition 1] OR [condition 2]----
             [condition N] ;
```

Eg:- Retriere ID, Name and salary from CUSTOMERS table where salary is greater than 2000 OR age less than 25 years

```
select ID, Name, salary FROM CUSTOMERS
WHERE salary > 2000 OR age<25;
```

ID	Name	Salary
3	Kaushik	2000
6	Komal	4500
7	Muffy	10000
4	chaitali	6500
5	Hardik	8500

ANY operator:-

The ANY operator compares a value with all the values returned by subquery and is true only if the given condition is satisfied for any value in the set of values.

Eg:- Retrieve the details of all CUSTOMERS whose salary is greater than the salary of even one customer of age 25.

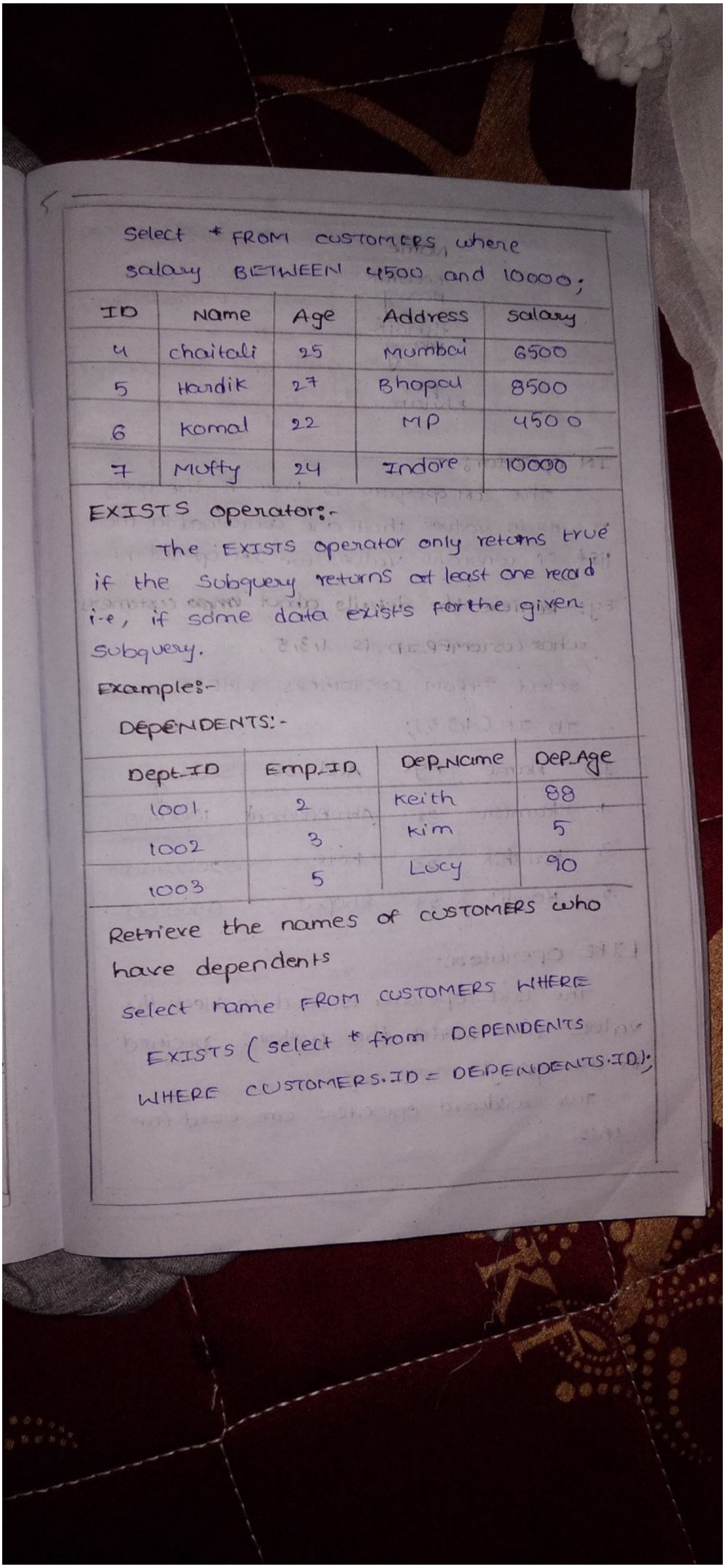
```
select * from CUSTOMERS WHERE
salary> ANY (select * FROM CUSTOMERS
WHERE age=25);
```

ID	Name	Age	Address	salary
3	kaushik	23	kota	2000
5	Hardik	27	Bhopal	8500
6	Komal	22	MP	4500
7	Muffy	24	Indore	10000

Between operator:-

The Between operator returns the information within a given range of values, where the minimum and maximum of the range is specified.

Eg:- Retrieve the details of CUSTOMERS whose salary is between the range of 4500 and 10000.



```
Select * FROM CUSTOMERS, where
salary BETWEEN 4500 and 10000;
```

ID	Name	Age	Address	salary
4	chaitali	25	Mumbai	6500
5	Hardik	27	Bhopal	8500
6	komal	22	MP	4500
7	Mufty	24	Indore	10000

EXISTS operator:-

The EXISTS operator only returns true if the Subquery returns at least one record i.e, if some data exists for the given subquery.

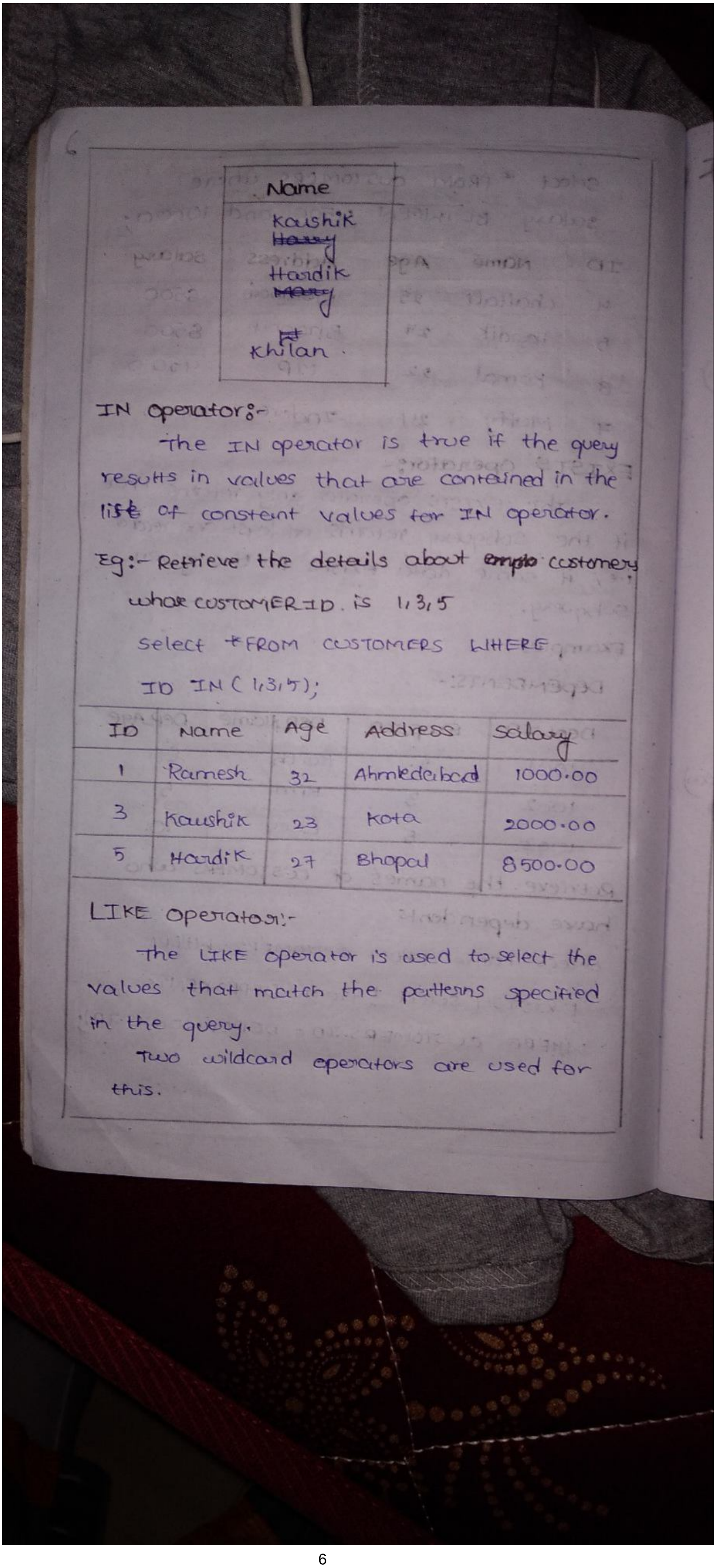
Examples:-

DEPENDENTS:-

Dept_ID	Emp_ID	Dep_Name	Dep_Age
1001	2	Keith	88
1002	3	Kim	5
1003	5	Lucy	90

Retrieve the names of CUSTOMERS who have dependents

```
select name FROM CUSTOMERS WHERE
 EXISTS ( select * from DEPENDENTS
WHERE CUSTOMERS.ID = DEPENDENTS.ID);
```



6

Name
Kaushik ~~Harry~~ Hardik ~~Mary~~ Khilan .

IN operator:-

The IN operator is true if the query results in values that are contained in the list of constant values for IN operator.

Eg:- Retrieve the details about ~~emplo~~ customer whoe CUSTOMERID. is 1,3,5

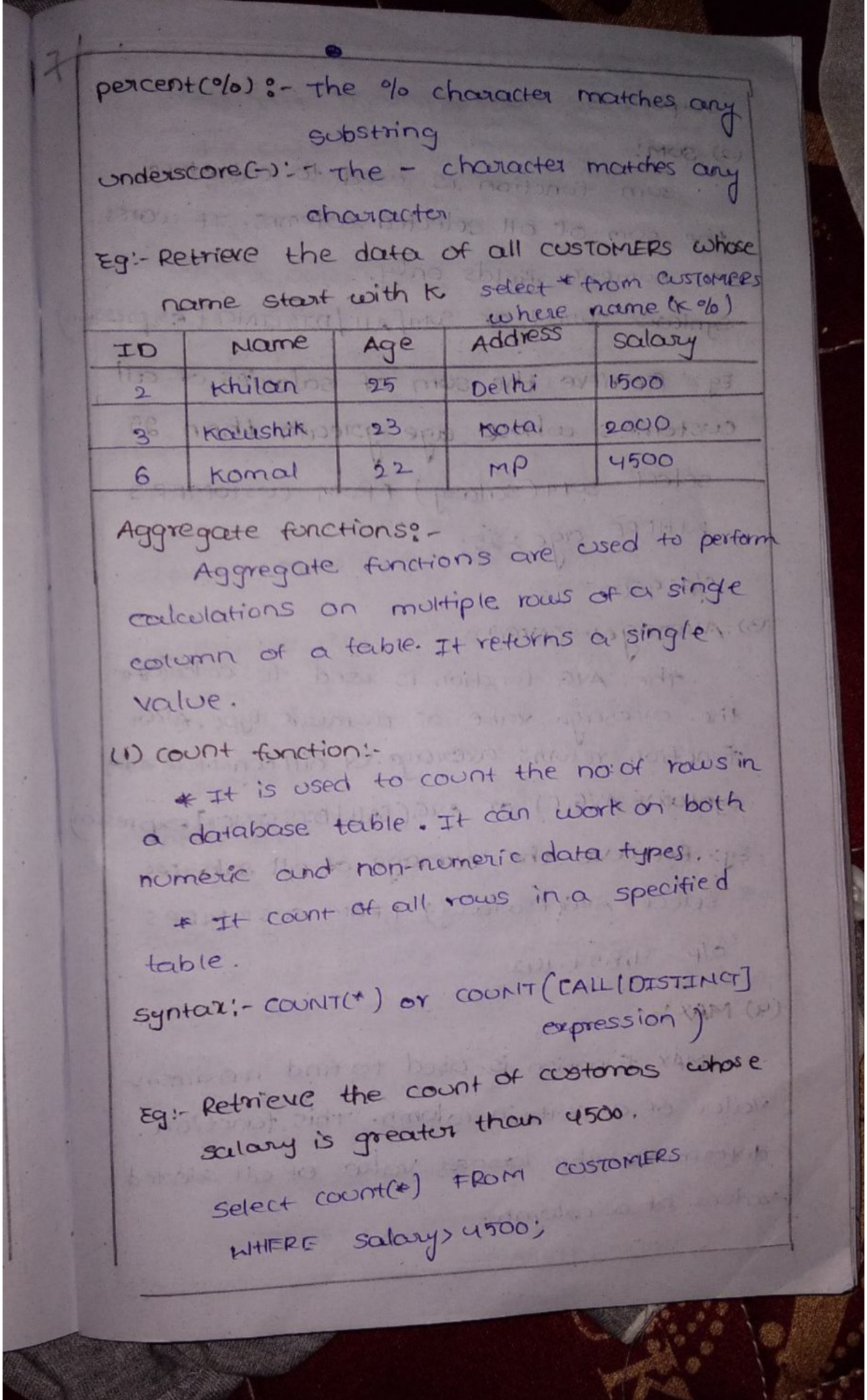
select *FROM CUSTOMERS WHERE ID IN(1,3,5);

ID	Name	Age	Address	Salary
1	Ramesh	32	Ahmledabad	1000.00
3	Kaushik	23	Kota	2000.00
5	Hardik	27	Bhopal	8500.00

LIKE operator:-

The LIKE operator is used to select the values that match the patterns specified in the query.

Two wildcard operators are used for this.



percent(%) :- The % character matches any substring

underscore(_):- The _ character matches any character

Eg:- Retrieve the data of all CUSTOMERS whose name start with K select * from CUSTOMERS where name (K %)

ID	Name	Age	Address	Salary
2	Khilan	25	Delhi	1500
3	Kaushik	23	Kota	2000
6	Komal	22	MP	4500

Aggregate functions:-

Aggregate functions are used to perform calculations on multiple rows of a single column of a table. It returns a single value.

(1) count function:-

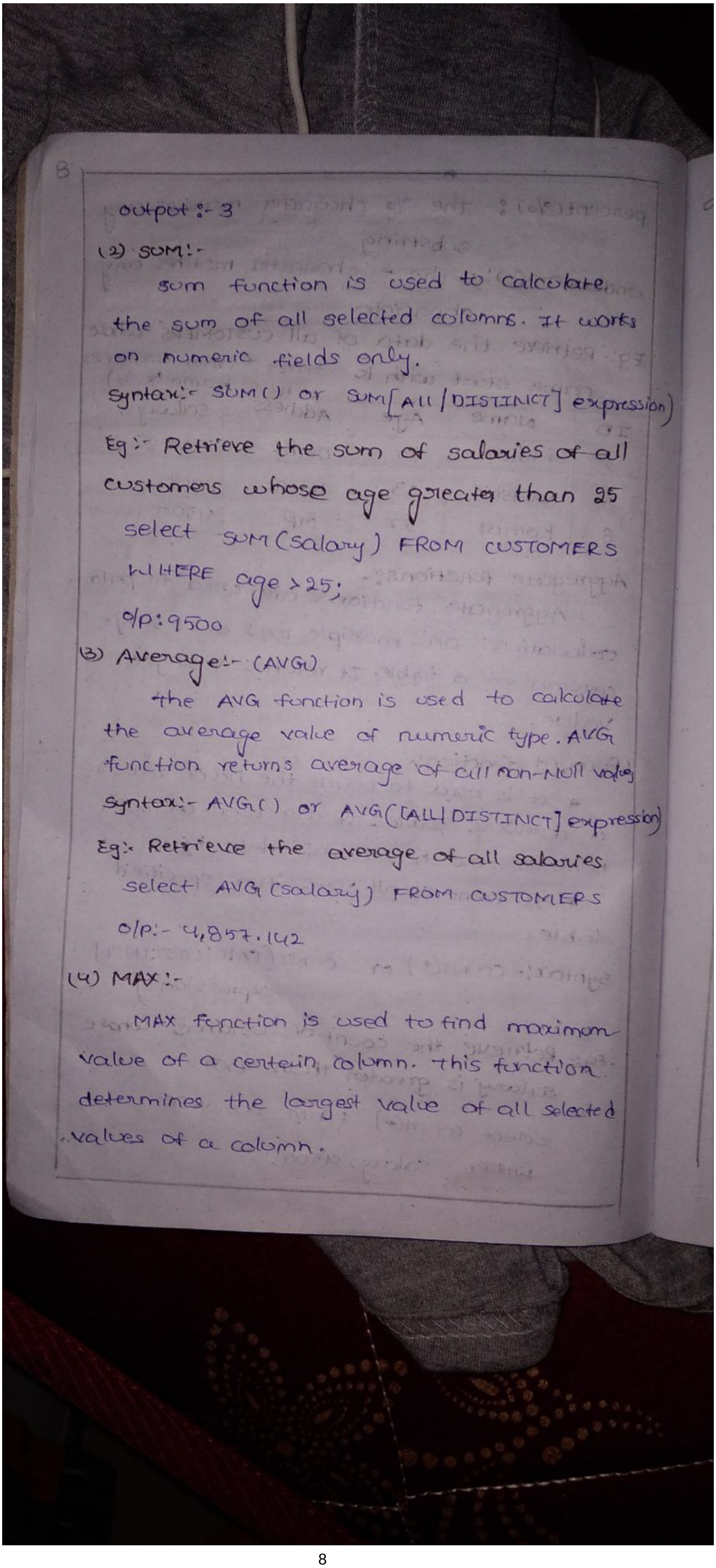
* It is used to count the no. of rows in a database table. It can work on both numeric and non-numeric data types.

* It count of all rows in a specified table.

Syntax:- COUNT(*) or COUNT([ALL|DISTINCT] expression)

Eg:- Retrieve the count of customers whose salary is greater than 4500.

select count(*) FROM CUSTOMERS WHERE salary> 4500;



output :- 3

(2) SUM:-

sum function is used to calculate the sum of all selected colomns. It works on numeric fields only.

syntax:- SUM() or SUM[All/DISTINCT] expression)

Eg:- Retrieve the sum of salaries of all customers whose age greater than 25

```
select SUM(salary) FROM CUSTOMERS
WHERE age >25;
```

o/p: 9500

(3) Average:- (AVG)

the AVG function is used to calculate the average vale of numeric type. AVG function returns average of all non-Null values

syntax:- AVG() or AVG([ALL| DISTINCT] expression)

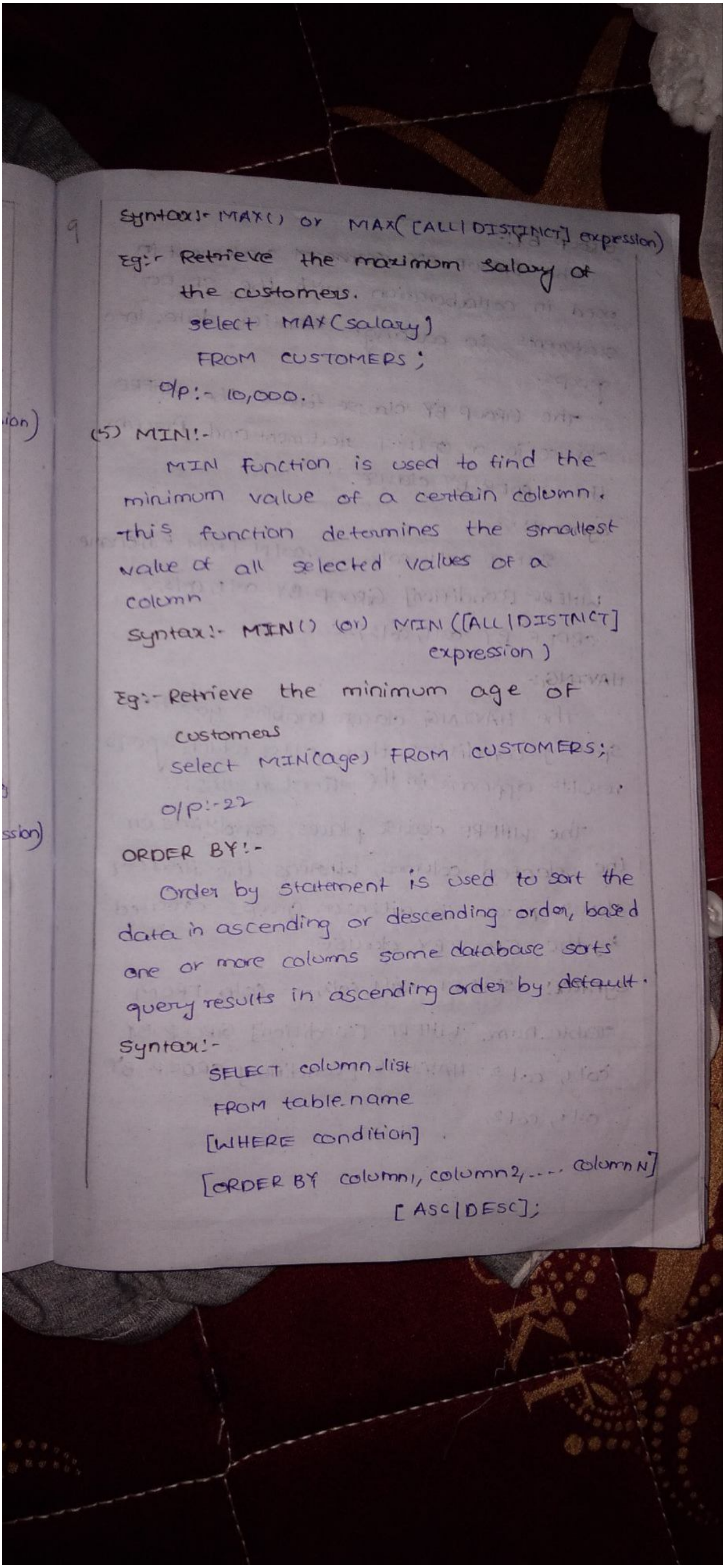
Eg:- Retrieve the average of all salaries

```
select AVG(salary) FROM CUSTOMERS
```

o/p:- 4,857.142

(4) MAX:-

MAX function is used to find maximum value of a certain column. this function determines the largest value of all selected values of a column.



Syntax:- MAX() or MAX([ALL|DISTINCT] expression)

Eg:- Retrieve the maximum salary of the customers.

```
select MAX(salary]
 FROM CUSTOMERS;
```

o/p:- 10,000.

(5) MIN:-

MIN function is used to find the minimum value of a certain column. this function determines the smallest value of all selected values of a column

Syntax:- MIN() (or) MIN ([ALL|DISTNCT] expression)

Eg:- Retrieve the minimum age of customers

```
select MIN(age) FROM CUSTOMERS;
```

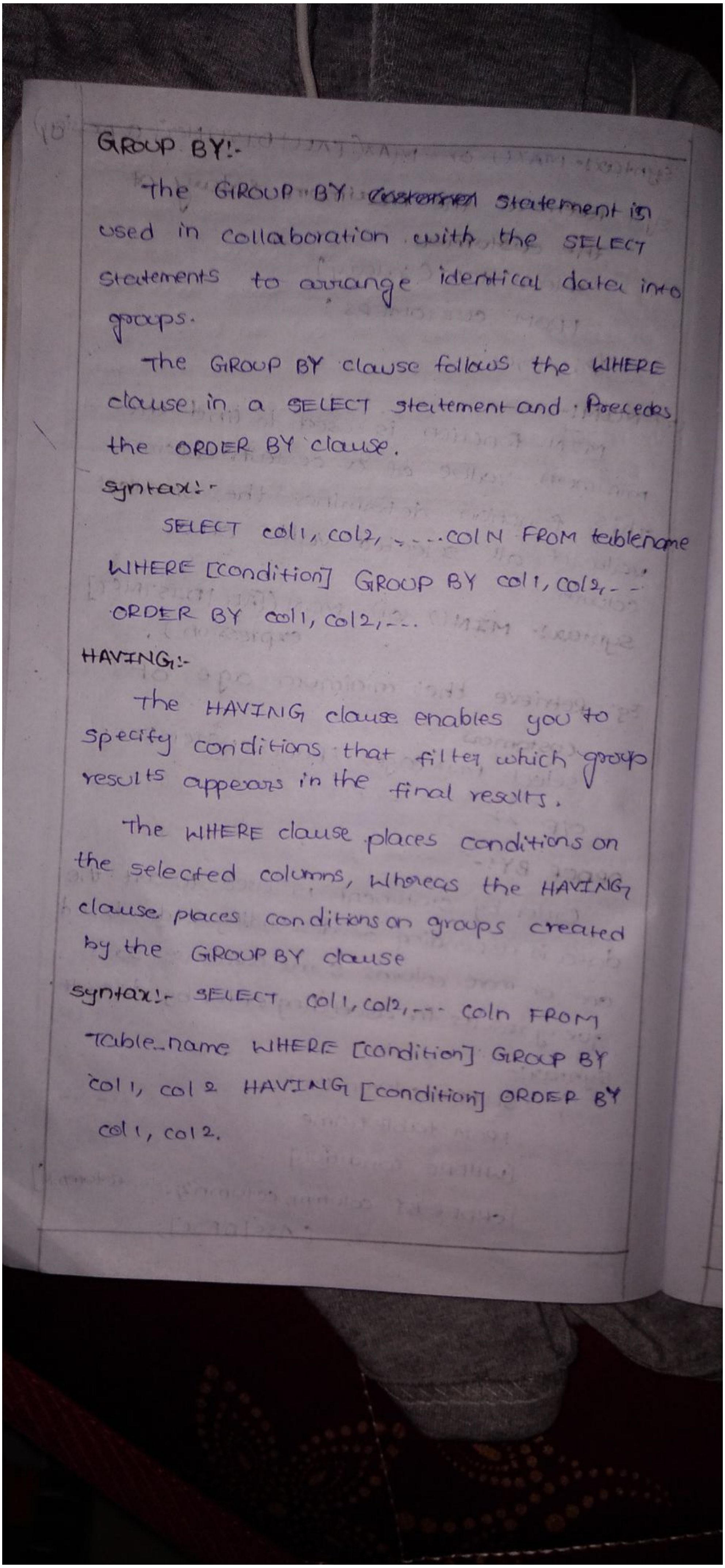
o/p:-22

ORDER BY:-

Order by statement is used to sort the data in ascending or descending order, based one or more colums some database sorts query results in ascending order by default.

Syntax:-

```
SELECT column_list
 FROM table.name
 [WHERE condition]
 [ORDER BY column1, column2,.... column N]
         [ ASC|DESC];
```



GROUP BY:-

The GROUP BY ~~customer~~ statement is used in collaboration with the SELECT statements to arrange identical data into groups.

The GROUP BY clause follows the WHERE clause in a SELECT statement and Precedes the ORDER BY clause.

syntax:-

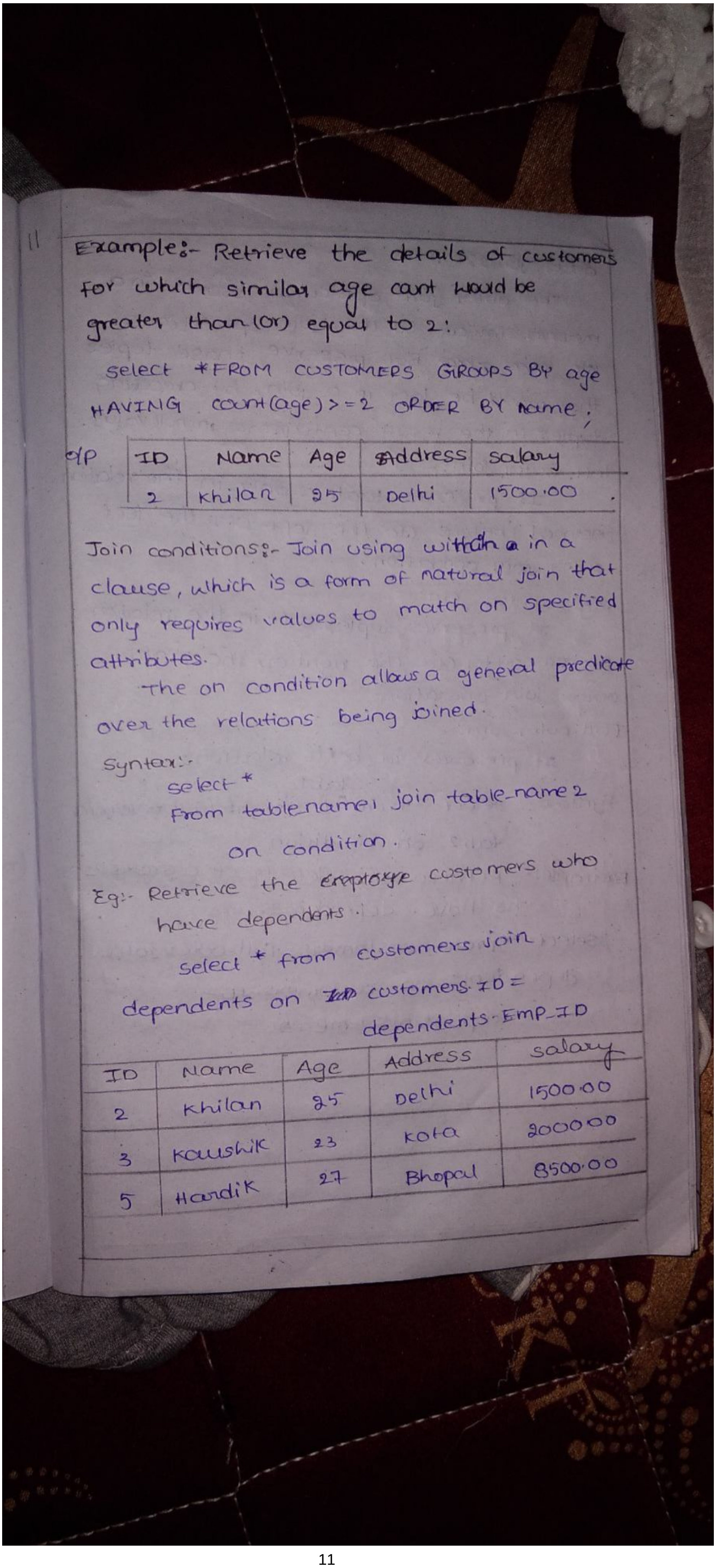
SELECT col1, col2, ...colN FROM tablename WHERE [condition] GROUP BY col1, col2,... ORDER BY col1, col2,...

HAVING:-

The HAVING clause enables you to specify conditions that filter which group results appears in the final results.

The WHERE clause places conditions on the selected columns, Whereas the HAVING clause places conditions on groups created by the GROUP BY clause

syntax:- SELECT col1, col2,... coln FROM Table_name WHERE [condition] GROUP BY col1, col2 HAVING [condition] ORDER BY col1, col2.



Example:- Retrieve the details of customers for which similar age count would be greater than (or) equal to 2:

```
select *FROM CUSTOMERS GROUPS BY age
HAVING count(age)>=2 ORDER BY name;
```

o/p

ID	Name	Age	~~$~~Address	salary
2	Khilan	25	Delhi	1500.00

Join conditions:- Join using with~~an~~ ~~a~~ in a clause, which is a form of natural join that only requires values to match on specified attributes.

The on condition allows a general predicate over the relations being joined.

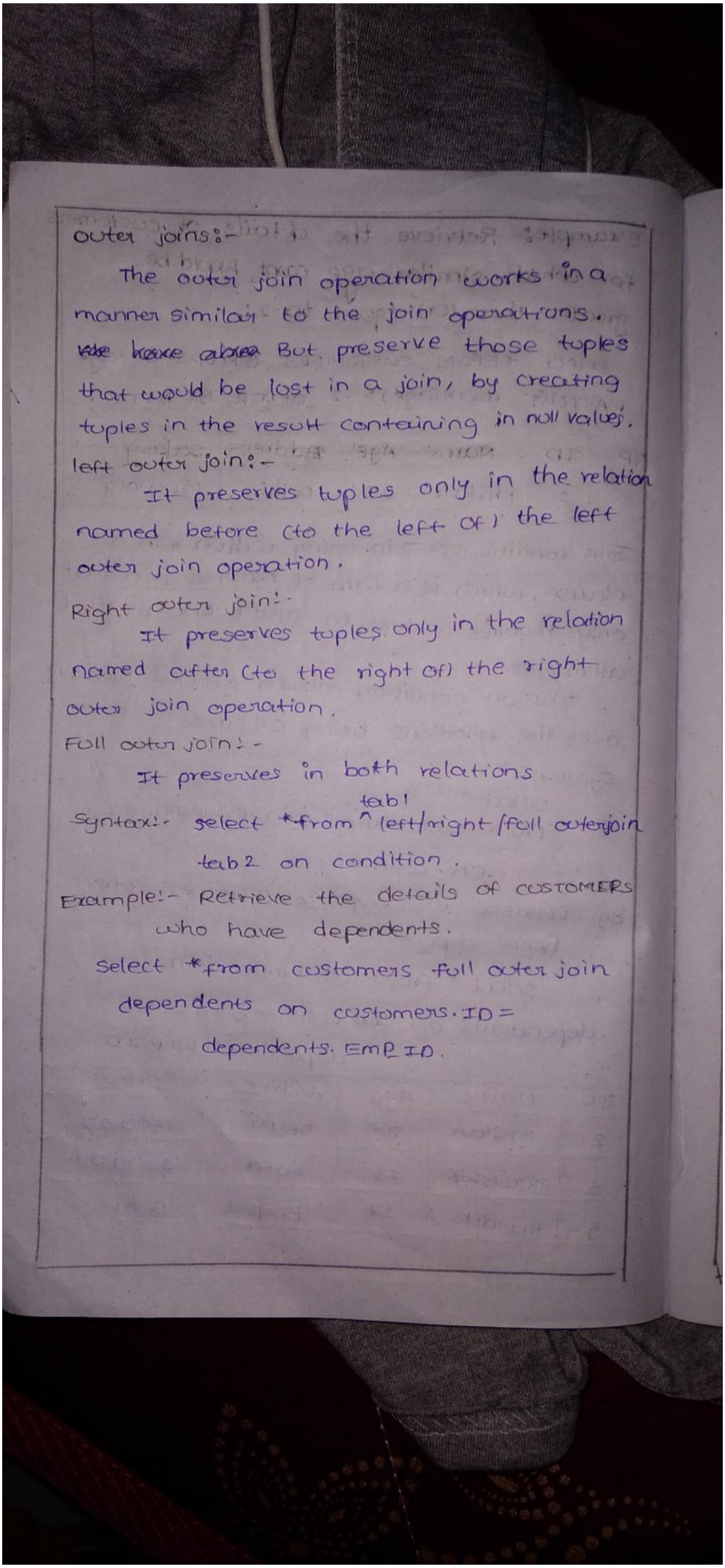
Syntax:-

```
select *
From tablename1 join table_name 2
        on condition.
```

Eg:- Retrieve the ~~employee~~ customers who have dependents.

```
select * from customers join
dependents on ~~ID~~ customers.ID =
                    dependents.EMP_ID
```

ID	Name	Age	Address	salary
2	Khilan	25	Delhi	1500.00
3	Kaushik	23	Kota	2000.00
5	Hardik	27	Bhopal	8500.00



outer joins:-

The outer join operation works in a manner similar to the join operations. ~~We have alrea~~ But preserve those tuples that would be lost in a join, by creating tuples in the result containing in noll values.

left outer join:-

It preserves tuples only in the relation named before (to the left of) the left outer join operation.

Right outer join:-

It preserves tuples only in the relation named after (to the right of) the right outer join operation.

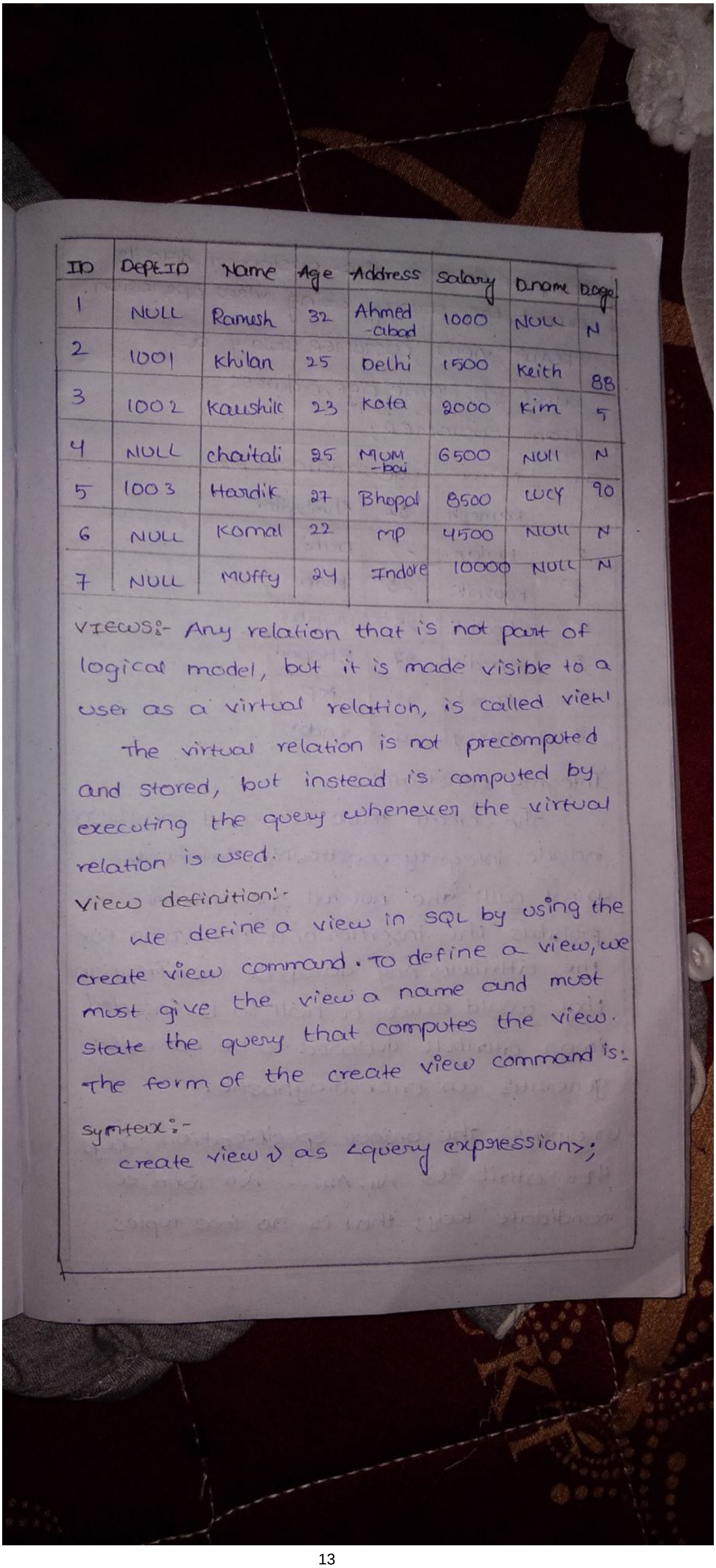
Full outer join:-

It preserves in both relations

Syntax:- select *from tab1 left/right/full outerjoin tab2 on condition.

Example:- Retrieve the details of CUSTOMERS who have dependents.

select *from customers full outer join dependents on customers.ID = dependents.EmP ID.



ID	DeptID	Name	Age	Address	Salary	Dname	Dage
1	NULL	Ramesh	32	Ahmed-abad	1000	NULL	N
2	1001	Khilan	25	Delhi	1500	Keith	88
3	1002	Kaushik	23	Kota	2000	Kim	5
4	NULL	chaitali	25	MUM-bai	6500	NUll	N
5	1003	Hardik	27	Bhopal	8500	LUCY	90
6	NULL	Komal	22	MP	4500	NULL	N
7	NULL	Muffy	24	Indore	10000	NULL	N

VIEWS:- Any relation that is not part of logical model, but it is made visible to a user as a virtual relation, is called view

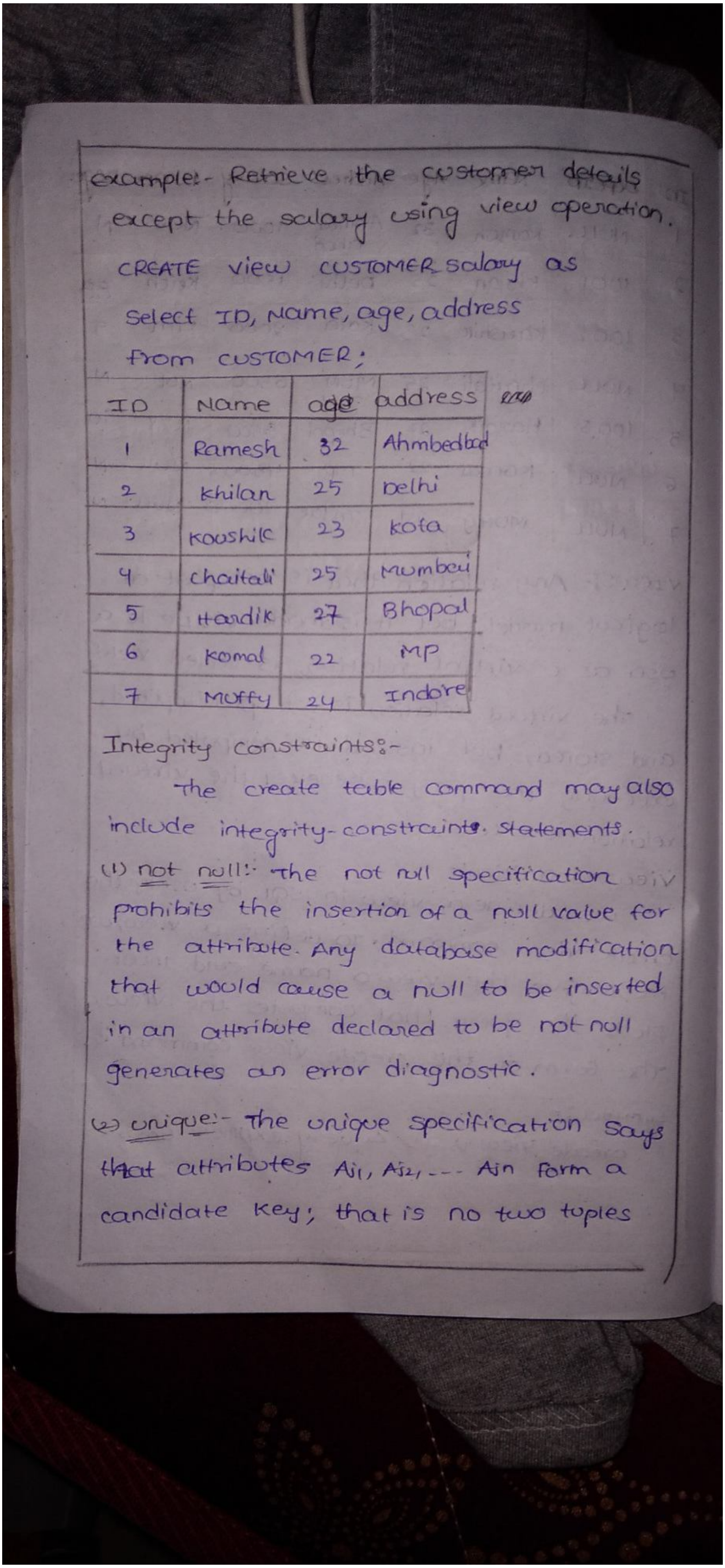
The virtual relation is not precomputed and stored, but instead is computed by executing the query whenever the virtual relation is used.

View definition:-

We define a view in SQL by using the create view command. To define a view, we must give the view a name and must state the query that computes the view. The form of the create view command is:

syntax:-

create view v as <query expression>;



example:- Retrieve the customer details except the salary using view operation.

```
CREATE view CUSTOMER salary as
Select ID, Name, age, address
from CUSTOMER;
```

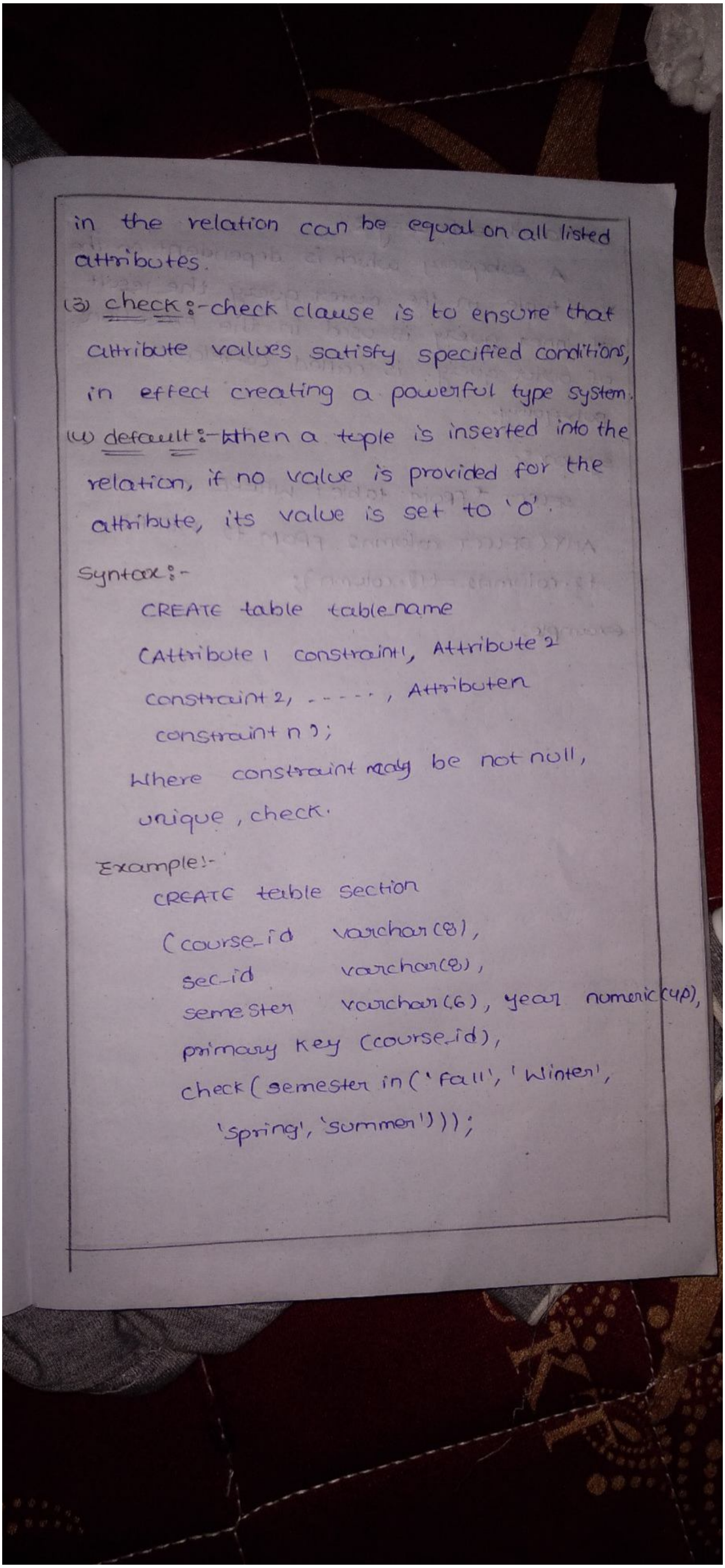
ID	Name	age	address
1	Ramesh	32	Ahmbedbad
2	khilan	25	Delhi
3	Koushik	23	kota
4	chaitali	25	Mumbai
5	Hardik	27	Bhopal
6	Komal	22	MP
7	Muffy	24	Indore

Integrity constraints:-

The create table command may also include integrity-constraints statements.

(1) not null:- The not null specification prohibits the insertion of a null value for the attribute. Any database modification that would cause a null to be inserted in an attribute declared to be not null generates an error diagnostic.

(2) unique:- The unique specification says that attributes $A_{i1}, A_{i2}, \ldots A_{in}$ form a candidate key; that is no two tuples



in the relation can be equal on all listed attributes.

(3) check :- check clause is to ensure that attribute values satisfy specified conditions, in effect creating a powerful type system.

(4) default :- When a tuple is inserted into the relation, if no value is provided for the attribute, its value is set to '0'.

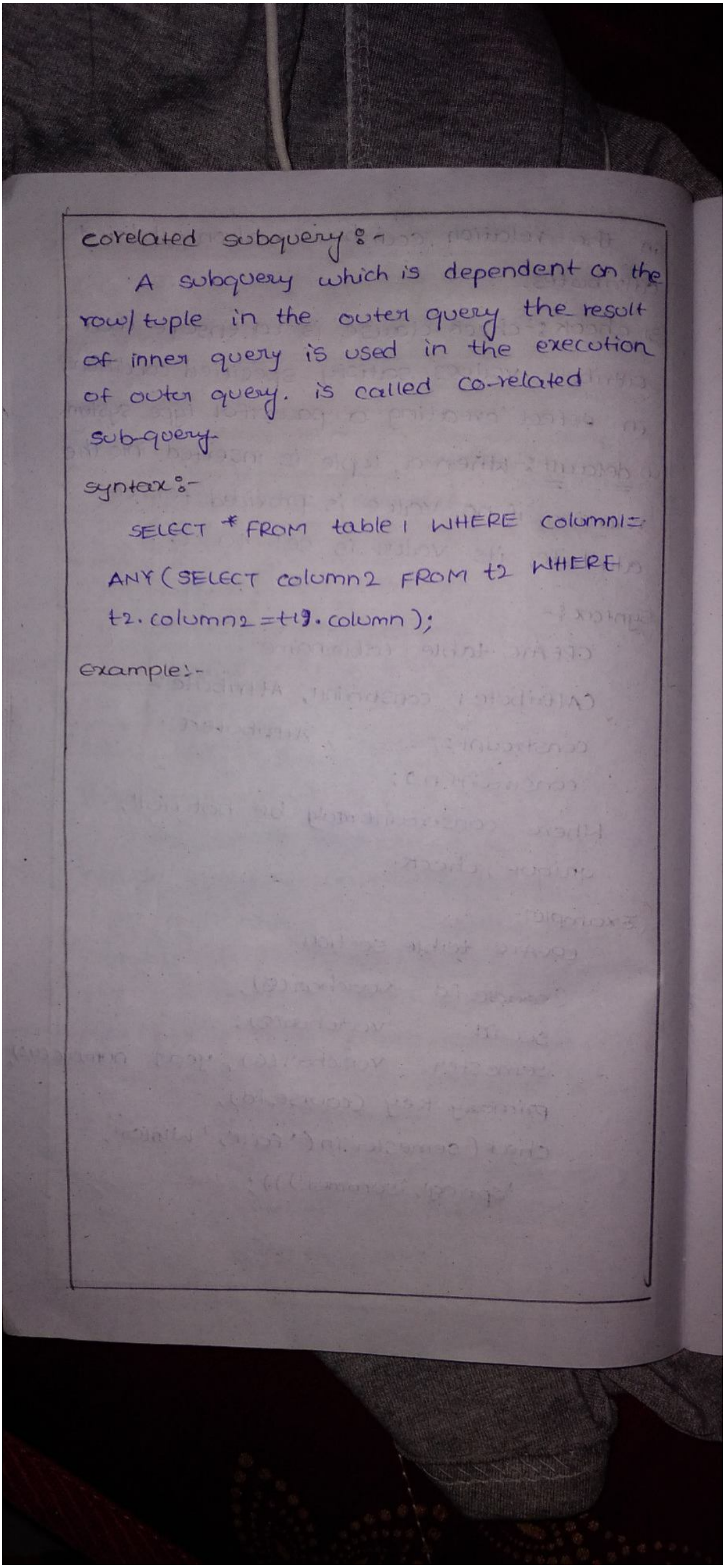
Syntax :-

```
CREATE table tablename
(Attribute 1 constraint1, Attribute 2
constraint 2, ----, Attributen
 constraint n );
```

Where constraint may be not null, unique, check.

Example:-

```
CREATE table section
 (course_id    varchar(8),
  sec_id       varchar(8),
  semester     varchar(6), year  numeric(4,0),
  primary key (course_id),
  check (semester in ('Fall', 'Winter',
    'Spring', 'Summer')));
```



corelated subquery :-

A subquery which is dependent on the row/tuple in the outer query the result of inner query is used in the execution of outer query. is called co-related sub-query.

syntax :-

```
SELECT * FROM table1 WHERE column1=
ANY (SELECT column2 FROM t2 WHERE
t2.column2=t19.column);
```

Example:-