PART-B

Subqueries

Sailors	SID, Sname, rating, age
Reserves	SID, BID, day
Boats	BID, Bname, color

* A query that has another query embedded within it.
* The embedded query is the subquery.

eg: Find the names of sailors who have reserved boat 103.

```
Select S.sname from Sailors S where S.sid in
( select R.sid from Reserves R where R.bid = 103)
```

* to find all the sailors who ~~don't~~ haven't reserved boat 103, just replace IN by NOT IN.

Multiple nested queries:-

Find the names of the sailors who have reserved a red boat.

```
Select S.sname from Sailors S where S.sid
① in ( Select R.sid from Reserves R where R.bid
② in ( select B.bid from Boats B where B.color = 'red' ))
```

* To find the names of sailors who have not reserved a red boat, we replace the outermost occurence of 'IN' by 'not in'

① Underlined in blue

might or might not have reserved other boats

* To find the names of the sailors who have reserved a boat that is not red, we replace the innermost occurence of 'in' with ~~'notin'~~ 'not in'

② Underlined

* If we change both the 'IN' with 'NOT IN', we get the names of sailors who have not ~~reserved~~ reserved a boat that is

not red - that is those who have reserved only red boats - if they have reserved any at all.

Correlated Subqueries :-

* In nested subqueries, the inner query is independent of the outer query.
* In correlated subqueries, the inner queries are dependent on the outer queries.

* eg: Find the names of sailors who have reserved boat number 103.

```
Select S.sname from Sailors S where exists
(select * from reserves R where R.bid = 103
and R.sid = S.sid)
```

* The 'exists' operator is another set comparison operator, such as 'IN'
* It allows us to test whether a set is nonempty.

* By using `not exists` instead of `exists`, we can compute the names of sailors who have not reserved ~~a red boat~~ boat no 103.

SQL Groupby statement :-

* The groupby statement groups rows that have the same values into summary rows, like "find the number of customers in each country".

* The groupby statement is often used with aggregate functions (count(), max(), min(), sum(), avg()) to group the result set by one or more columns.

Syntax :-

```
Select column_name(s) from table_name
groupby column_name(s) order by
column_name(s);
```

eg:

Tablename : Customers

Columnnames : CustomerID, CustomerName, ContactName, Address, City, Postalcode, country.

=> To find the number of customers in each country.

```
Select count(customerID), country
  from customers groupby country;
```

=> To find the number of customers in each country sorted high to low:

```
Select count(customerID), country
  from customers groupby country
  orderby count(customerID) DESC;
```

SQL Having clause

* The 'having' clause was added to SQL because the 'where' keyword cannot be used with aggregate functions.

Syntax:

Select column_name(s) from table_name
groupby column_name(s) having condition
order by column_name(s);

eg: (same table)

=> To list the number of customers in each country and to only include countries with more than 5 customers.

Select count(customerID), country
from customers groupby country
having count(customerID) > 5;

=> The number of customers in each country sorted high to low (only include countries with more than 5 customers):

Select count(customerID), country
from customers group by country
having count(customerID) > 5
orderby count(customerID) DESC;

SQL orderby Clause :-

* The order by command is used to sort the result set in ascending or descending order.
* The orderby command sorts the result set in ascending order by default.
* To sort the records in descending order, use the desc keyword.

eg:- To select all the columns from the customers table, sorted by the 'customername' column

```
Select * from customers
order by customername;
```

ASC

* The ASC command is used to sort the data returned in ascending order.

eg: To select all the ~~customers~~ columns from the customers table sorted by the customername column.

```
Select * from customers
orderby customername ASC;
```

DESC :-

* The DESC command is used to sort the data returned in descending order.

eg:- To select all the columns from the "customers" table sorted descending by the "customername" column:

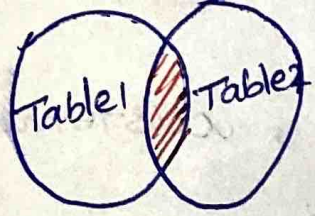
```
Select * from customers
order by customername desc;
```

SQL Joins

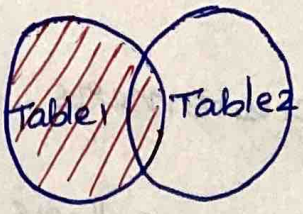
* A join clause is used to combine rows from two or more tables, based on a related column between them.

Different types of SQL joins

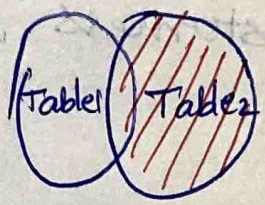
* ~~join~~ (Inner) join
* left (outer) join
* right (outer) join
* full (outer) join.



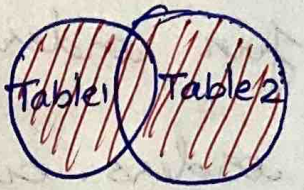
Inner Join



Left Join



Right Join



Full Outer Join

~~Inner~~ Join :-

* The inner join keyword selects records that have matching values in both tables.
* ~~Just~~ join, ~~also~~ gives the same result. al inner join

Syntax :-

```
Select column_name(s) from table1
~~inner~~ join table2 on table1.column_name =
table2.column_name;
```

* It is a good practice to include the table name when specifying columns in the SQL statement.

tablename.column_name

eg:-

```
Select customers.customer_id, customers.
first_name, orders.amount from customers
join orders on customers.customer_id =
orders.customer;
```

Table: customers

customer_id	First_name
1	John
2	Robert
3	David
4	John
5	Betty

Table: Orders

Order_id	amount	custom
1	200	10
2	500	3
3	300	6
4	800	5
5	150	8

Customer_id	first_name	amount
3	David	500
5	Betty	800

Inner Join

* Join and inner join will return the same result.

* Inner is the default join & type for Join, so when you write join, the parser actually writes inner join.

* It returns records that have matching values in both the tables.

eg: Select customers.customer_id, customers.first_name, orders.amount from customers inner join orders on customers.customer_id = orders.customer;

o/p

customer_id	first_name	amount
3	David	500
5	Betty	800

Left Join :-

* The left join keyword returns all records from the left table (table 1) and the matching records from the right table (table 2).
* The result is 0 records from the right side if there is no match.

Syntax:

Select column_name(s) from table1 left join table2 on table1.column_name = table2.column_name

eg: Select customers.customer_id, Customers.first_name, orders.amount from customers left join orders on customers.customer_id = Orders.customer;

op

customer_id	first_name	amount
1	John	
2	Robert	
3	David	500
4	John	
5	Betty	800

Right Join :-

* The right join keyword returns all records from the right table (table 2), and the matching records from the left table (table 1).
* The result is 0 records from the left side, if there is no match.

Syntax:

```
Select column_name(s) from table1 right join
table2 on table1.column_name = table2.
column_name ;
```

eg:-

```
Select customers.customer_id, customers.
first_name, orders.amount from customers
right join orders on customers.customer_id=
orders.customer ;
```

customer_id	first_name	amount
3	David	600
5	Betty	800
		200
		300
		150

Full Outer Join :-

* The full outer join keyword returns all records when there is a match in left (table 1) or right (table 2) table records.
* Full outer join and full join are the same.
* Returns all records when there is a match in either left or right table.

Syntax: select column_name(s) from table1 full outer join table2 on table1.column_name = table2.column_name where condition;

eg:

```
select customers.customer_id, customers.
first_name, orders.amount from customers
full outer join orders on customers.
customer_id = orders.customer;
```

dp:-

customer_id	first_name	amount
3	David	200
		500
5	Betty	300
		800
1	John	150
2	Robert	
4	John	

Exists keyword

* The exists operator is used to test for the existence of any record in a subquery.
* The exists operator returns true if the subquery returns one or more records.

Syntax :-

```
select column_name(s) from table_name
where exists (select column_name from
table_name where condition);
```

Demo database.

Products (ProductId, ProductName, SupplierID, CategoryID, Unit, Price)

Suppliers (SupplierID, SupplierName, ContactName, Address, City, PostalCode, Country)

Examples :-

* SQL statement which lists the suppliers with a product price less than 20 and returns true.

↳ Select SupplierName from Suppliers where Exists (Select ProductName from Products where Products.SupplierID = Suppliers.SupplierID AND Price < 20);

* SQL statement which lists the suppliers with a product price equal to 22 and returns true.

L> Select SupplierName From Suppliers
Where exists (Select ProductName
from Products where Products.SupplierID
= Suppliers.SupplierID and price = 22);

SQL Any Operator

* SQL 'any' compares a value of the first table with all values of the second table and returns the row if there is a match with any value.

Syntax:

Select column from table1 where column
operator any (select column
from table 2);

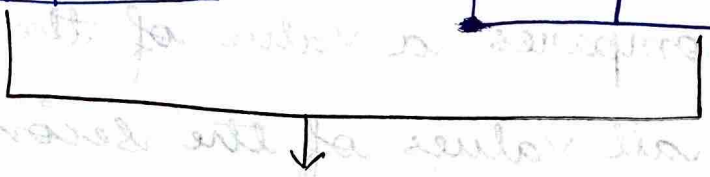
eg: Query : Select * from teachers
where age = any (select age
from students);

Table: Teachers

ID	Name	Age
1	Peter	32
2	Megan	43
3	Rose	29
4	Linda	30
5	Mary	41

Table: Students

ID	Name	Age
1	Harry	23
2	Jack	42
3	Joe	32
4	Dent	23
5	Bruce	40



ID	Name	Age
1	Peter	32

SQL Any with < operator :-

* We can use any comparison operators like =, >, < etc with the any and all keywords.

eg: select * from teachers where age < any (select age from students);

o/p

ID	Name	Age
1	Peter	32
3	Rose	29
4	Linda	30
5	Mary	41

SQL All Operator :-

* SQL all compares a value of the first table with all values of the second table and returns the row if there is a match with all values.

Syntax :-

Select column from table1 where column operator all (select column from table 2);

eg: Select * from teachers where age > all (select age from students);

o/p :-

ID	Name	Age
2	Megan	43

Transaction Control Commands

* Transactions group a set of tasks into a single execution unit.
* Each transaction begins with a specific job and ends when all the tasks in the group are successfully completed.
* If any of the tasks fail, the transaction fails.
* Therefore, a transaction has only 2 results Success or failure
* Incomplete steps result in the failure of the transaction.

ACID

* A database transaction by defn must be ACID.

A - Atomic, C - consistent, I - Isolated, D - Durable.

* These properties can ensure the concurrent execution of multiple transactions without conflict.

Atomicity - The outcome of the transaction can either be completely successful or completely unsuccessful.

* The whole transaction must be rolled back if part of it fails.

Consistency - Transactions maintain integrity restrictions by moving the database from one valid state to another.

Isolation - Concurrent transactions are isolated from one another, assuring the accuracy of the data.

Durability :- Once a transaction is committed, its modifications remain in effect even in the event of a system failure.

3 Main commands :-

→ Commit
→ Rollback
→ Savepoint

Commit Command

* If everything is in order with all statements within a single transaction, all changes are recorded together in the database is called committed.
* The commit command saves all the transactions to the database since the last commit or rollback command.

Syntax : Commit ;

Rollback command

* If any error occurs with any of the SQL grouped statements, all changes need to be aborted.
* The process of reversing changes is called rollback.
* This command can only be used to undo transactions since the last commit or rollback command was issu

Syntax : Rollback ;

Savepoint command.

* Savepoint creates points within the groups of transactions in which to rollback.
* A savepoint is a point in a transaction in which you can roll the transaction back to a certain point without rolling back the entire transaction.

Syntax to create savepoint :-

savepoint savepoint_name ;

Syntax to rollback to a savepoint command :-

rollback to savepoint_name ;

Release savepoint :-

* This command is used to remove a savepoint that you have created.

Syntax : release savepoint savepoint_name ;

* Once a savepoint has been released, you can no longer use the rollback command to undo transactions performed since the last savepoint.

Cursors :-

* Cursor is a temporary memory or temporary work station.
* It is allocated by database server at the time of performing DML operations on table by user.
* Cursors are used to store database tables.
* There are 2 types of cursors:
  -> Implicit cursors
  -> Explicit cursors.

Implicit Cursors

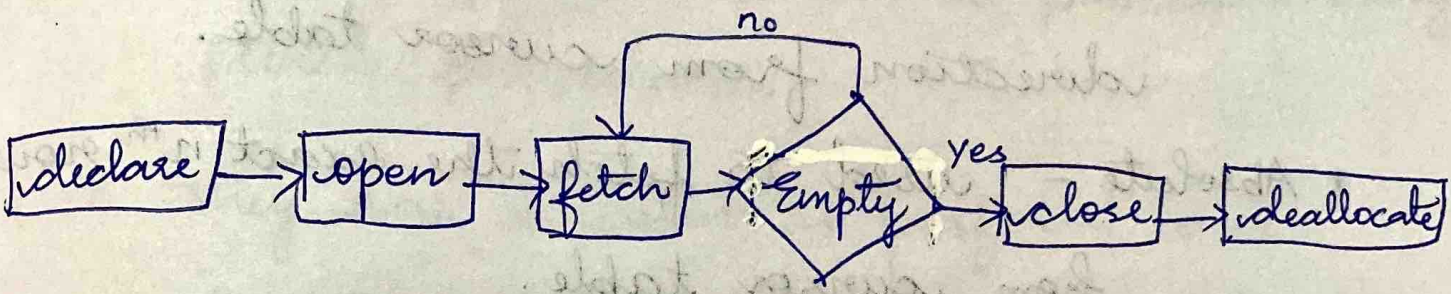
* Also known as default cursors of SQL server.
* They are allocated by the SQL server when the user performs DML operations

Explicit Cursors.

* Explicit cursors are created by users whenever the user requires them.

* Explicit cursors are used for fetching data from table in row-by-row manner.

SQL cursor life cycle :-



How to create explicit cursor:

```
// declare a cursor
declare cursor_name cursor for select_statement;
// open
open cursor_name;
// fetch
begin fetch next from cursor_name;
end;
```

* First - used to fetch only the 1st row from cursor table.
* Last - used to fetch only last row from cursor table.

* Next - used to fetch data in forward direction from cursor table.

* Prior - used to fetch data in backward direction from cursor table.

* Absolute - used to fetch the exact n^{th} row from cursor table.

=> To close cursor connection

```
close cursor_name;
```

=> To deallocate cursor memory

```
deallocate cursor_name;
```

Stored Procedure :-

* A stored procedure in SQL is a type of code in SQL that can be used for stored for later use and can be used many times.

* Whenever you need to execute the query, instead of calling it, you can just call the stored procedure.

* Values or parameters can be passed through stored procedures.

* They are created to perform one or more DML operations on database.

Syntax :- CREATING

create procedure procedure_name as
sql_statement go;

EXECUTING - Exec procedure_name

Triggers :-

* A trigger is a stored procedure in database which automatically invokes whenever a special event in the database occurs.

* eg: a trigger can be invoked when a row is inserted into a specific table or when certain table columns are being updated.

Syntax :-

create trigger triggername ① before/after ② insert/update/delete ③ on table_name ④ foreach ⑤ trigger_body ; ⑥

1 - creates or replaces an existing trigger with the trigger name.

2 - This specifies when the trigger will be executed.

3 - This specifies the DML operation.

4 - This specifis the name of the table associated with the trigger

5 - This specifies a row level trigger, (i.e) the trigger will be executed for each row being affected.

6 - This provides the operation to be performed as trigger is fired.