DAY 2 SQL - Learning

Transactions => series of operations seen as a whole.

Transfer Amount from 'A' to 'B'

for ex:

① A => deduct amount bal -= Amount

② B => credit amount bal += Amount

many/whole operation seen as one Transaction

ACID PROPERTIES:

- Atomicity
- Consistency
- Isolation
- Durability

Atomicity :-

All statements succeed or none succeed.

Consistency :-

Data moves from one valid state another

Isolation :-

Parallel transactions dont interfere

Durability :

committed data is permanently saved.

`SET autocommit = 0;`	disable autocommit during transaction
`SET autocommit = 1;`	enable autocommit after successful transaction.

To check the state of autocommit

```
SELECT @@autocommit;
```

TRANSACTIONS

Start & Commit

syntax

```
START TRANSACTION;
UPDATE accounts SET balance = balance - 50 where id = 1;
UPDATE accounts SET balance = balance + 50 where id = 2;
COMMIT;
```

ROLLback

PAGE NO. :
DATE : / /

* ROLLback brings the database back to its previous safe state
* ROLLback is an SQL command used to undo changes in transaction before they are permanently saved.

SYNTAX

```
ROLLBACK;
```

```
START TRANSACTION;
done
{ UPDATE accounts SET balance = balance + 100
                                    WHERE id = 1';
  COMMIT;
{ UPDATE accounts SET balance = balance - 100 WHERE
                                    id = 3;
  ROLLBACK;
not done
```

Savepoints : checkpoint inside a transaction.

SAVEPOINT is an SQL command used to mark a point inside a transaction so that you can roll back only part of the transaction not everything.

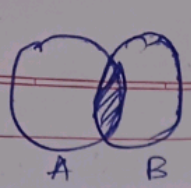
Sreema v

SYNTAX : SAVEPOINT savepoint_name;

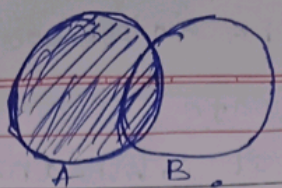
Daisy

SYNTAX : ROLLBACK TO savepoint_name;

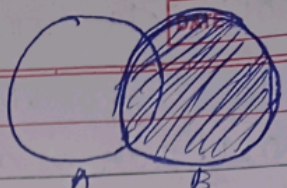
JOINS



Inner Join

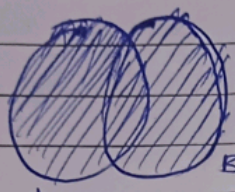


Left join



Right Join

Outer joins



Full join

outer join.

etc...

Inner join (Intersection)

```
SELECT column(s)
FROM tableA
INNER JOIN tableB
ON tableA.col_name = tableB.col_name;
```

Example 1

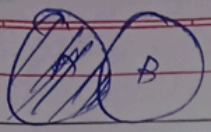
```
SELECT * FROM customer_table C
INNER JOIN Orders_table O
ON C.customer_id = O.customer_id;
```

EXAMPLE 2

```
SELECT c.name, O.order_id
FROM customer_table C
INNER JOIN Order_table O
ON C.customer_id = O.customer_id;
```

Left JOIN

only A ~~and~~ or fully A



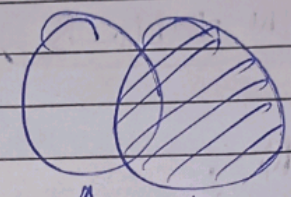
SYNTAX:

```
SELECT columns
FROM table A
LEFT JOIN table B
ON table A.col-name = table B.col-name;
```

if for value in table A doesn't hold any information in table be then filled with NULL

RIGHT JOIN

only B

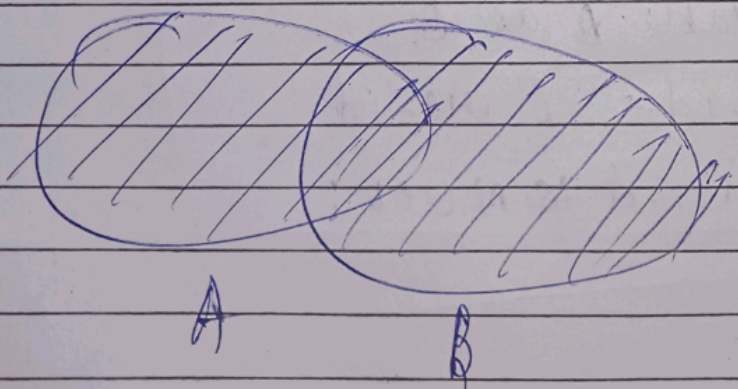


SYNTAX

```
SELECT columns
FROM table A
RIGHT JOIN table B
ON tableA.cus-id = tableB.cus_id;
```

if a value in "B" doesn't hold any information from "A" then NULL.

(UNION) Left join & Right join → Outer join



Syntax:

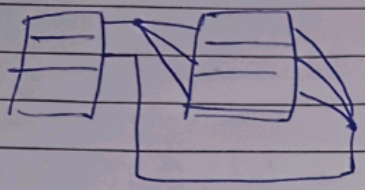
```
LEFT JOIN
UNION
RIGHT JOIN;
```

CROSS JOIN

Looks same as Outer join But completely different

PAGE NO. :
DATE : / /

A B



Every data in "A" access all the data in "B"

SYNTAX

```
SELECT * FROM table_A
CROSS JOIN table_B;
```

SELF JOIN :

A JOINS A

SYNTAX :

```
SELECT * FROM table_A as A
JOIN table_A as B
ON A.cust_name = B.cust_name;
```

Exclusive joins

Left_Exclusive join :-

```
Select * from table A as A
Left join table B as B
ON A.cus_id = B.cus_id
where B.cus_id is NULL;
```

Right Exclusive join

PAGE NO.:
DATE: / /

SYNTAX

```
Select * from table_A as A
RIGHT JOIN table_B as B
ON A.cus_id = B.cus_id
WHERE A.cus_id IS NULL;
```

Sub Queries:

A sub-Query or Inner query or a Nested query is a query within another SQL Query. It involves a select statements

SYNTAX:

```
SELECT COLUMN(S)
FROM table_name
WHERE col-name operator
(subquery);
```

USING Where

SYNTAX

```
SELECT *
FROM table_name
WHERE amount >
  ( SELECT amount * AVG(amount)
                    FROM table_nam
```

SYNTAX

```
SELECT * FROM table_name
Where col_name > val_name
    ( SELECT AVG(amount)
      FROM table_name);
```

USING SELECT

SYNTAX

```
SELECT name,
(SELECT COUNT(*)
   FROM table_name1
   Where table_name1.col_name = table_name2.col_name)
   AS select_name
   From customer c;
```

USING FROM

```
SELECT
summary.customer_id,
summary.avg_amount
FROM
( SELECT customer_id, AVG(AMOUNT) as avg_amou
   FROM orders
   GROUP BY customer_id
)
   AS summary;
```

VIEWS IN SQL

check out my code / query to understand better

A view is a virtual table based on the result set of an SQL statement.

A view always shows up-to-date data, the DB engine recreates the view every time the user queries it.

SYNTAX

```
CREATE VIEW views AS
SELECT col1, col2 FROM table_name;
```

```
DROP VIEW view_name;
```

USING JOIN

```
CREATE VIEW view_name AS
SELECT c.col1, c.col2, c.col3
FROM customer c
INNER JOIN Orders O
ON c.col1 = o.col1;
```

* NO Data stored physically (unless some Materialized view in some DBs)
* can include columns from one or more tables.
* can be used in
  - SELECT
  - JOIN
  - WHERE like a normal table
* Helps with security by exposing only certain columns to users.

INDEX IN SQL

PAGE NO.:
DATE: / /

Indexes are special database objects that make data retrival faster.

SYNTAX SINGLE COLUMN:

```
CREATE INDEX idx_name ON table(cols);
```

SYNTAX MULTI COLUMN

```
CREATE INDEX idx_name ON table(col1, col2);
```

```
SHOW INDEX FROM table;
```

~~DROP INDEX FROM tab~~

```
DROP INDEX idx_name ON table;
```

STORED PROCEDURES:

Predefined set of SQL statements that you can save in the database and execute whenever needed.

SYNTAX CREATE

```
CREATE PROCEDURE name(parameters)
BEGIN
    -- statement
END;
```

CALL

PAGE NO. :
DATE : /

```
CALL PROCEDURE pro_name;
```

DROP

```
DROP PROCEDURE Pro-name;
```