Transactions

ACID Properties

- **Atomicity** - All statements succeed or none succeed.
- **Consistency** - Data moves from one valid state to another.
- **Isolation** - Parallel transactions don't interfere.
- **Durability** - Committed data is permanently saved.

Transactions

Disable autocommit

```
SET autocommit = 0;
```

Enable autocommit

```
SET autocommit = 1;
```

Transactions

Start & Commit

```
START TRANSACTION;

UPDATE accounts SET balance = balance - 50 WHERE id = 1;
UPDATE accounts SET balance = balance + 50 WHERE id = 2;

COMMIT;
```

Transactions

Rollback

```
START TRANSACTION;

UPDATE accounts SET balance = balance - 100 WHERE id = 1;
UPDATE accounts SET balance = balance + 100 WHERE id = 3;

ROLLBACK;
```

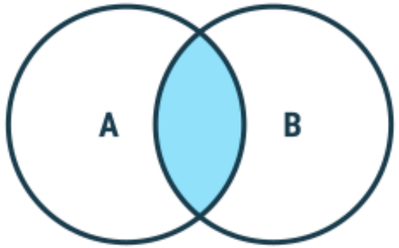
Transactions

Savepoint

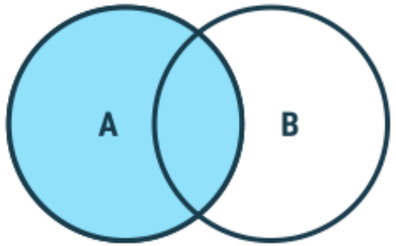
```
START TRANSACTION;

UPDATE accounts SET balance = balance + 1000 WHERE id = 1;
SAVEPOINT after_wallet_topup;

UPDATE accounts SET balance = balance + 10 WHERE id = 1;

ROLLBACK TO after_wallet_topup;

COMMIT;
```

JOINs

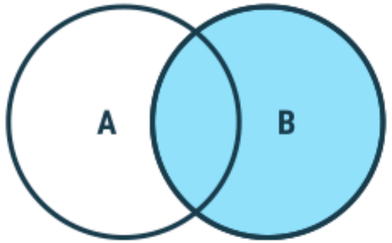
JOINs are used to combine rows from two or more tables based on a related column between them.



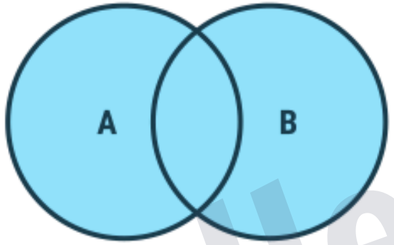
Inner Join



Left Join



Right Join

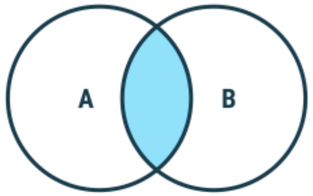


Full Join

Outer Joins

JOINs

INNER JOIN



customer_id	name	city
1	Alice	Mumbai
2	Bob	Delhi
3	Charlie	Bangalore
4	David	Mumbai

customers

order_id	customer_id	amount
101	1	500
102	1	900
103	2	300
104	5	700

orders

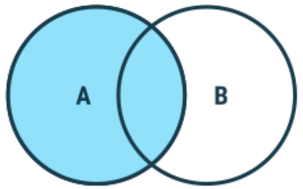
Syntax

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

```sql
-- inner join
SELECT c.name, o.order_id, o.amount
FROM customers c
INNER JOIN orders o
ON c.customer_id = o.customer_id;
```

JOINs

LEFT JOIN



customer_id	name	city
1	Alice	Mumbai
2	Bob	Delhi
3	Charlie	Bangalore
4	David	Mumbai

customers

order_id	customer_id	amount
101	1	500
102	1	900
103	2	300
104	5	700

orders

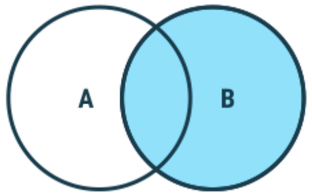
Syntax

SELECT *column(s)*
FROM ***tableA***
LEFT JOIN ***tableB***
ON *tableA.col_name = tableB.col_name;*

```sql
-- left join
SELECT *
FROM customers c
LEFT JOIN orders o
ON c.customer_id = o.customer_id;
```

JOINs

RIGHT JOIN



customer_id	name	city
1	Alice	Mumbai
2	Bob	Delhi
3	Charlie	Bangalore
4	David	Mumbai

customers

order_id	customer_id	amount
101	1	500
102	1	900
103	2	300
104	5	700

orders

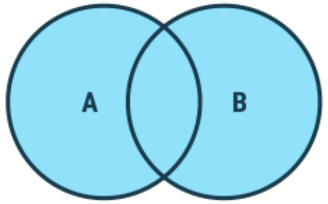
Syntax

SELECT *column(s)*
FROM ***tableA***
RIGHT JOIN ***tableB***
ON *tableA.col_name = tableB.col_name;*

```sql
-- right join
SELECT *
FROM customers c
RIGHT JOIN orders o
ON c.customer_id = o.customer_id;
```

JOINs

OUTER JOIN



LEFT JOIN

UNION

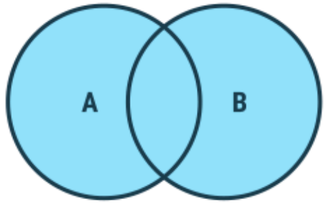
RIGHT JOIN

Syntax in MySQL

```
SELECT * FROM customers as c
LEFT JOIN orders as o
ON c.customer_id = o.customer_id
UNION
SELECT * FROM customers as c
RIGHT JOIN orders as o
ON c.customer_id = o.customer_id;
```

JOINs

CROSS JOIN



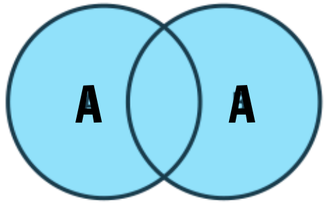
Syntax

SELECT *column(s)*
FROM ***tableA***
CROSS JOIN ***tableB*** ;

```sql
-- cross join
SELECT *
FROM customers as c
CROSS JOIN orders as o;

-- inner join
SELECT *
FROM customers as A
JOIN customers as B
ON A.customer_id = B.customer_id;
```

JOINs

SELF JOIN



It is a regular join but the table is joined with itself.

Syntax

SELECT *column(s)*
FROM ***table as a***
JOIN ***table as b***
ON *a.col_name = b.col_name;*

Practice Qs

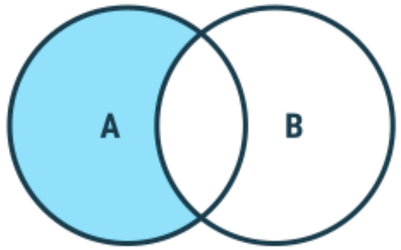
Write SQL command to display the exclusive joins :

customer_id	name	city
1	Alice	Mumbai
2	Bob	Delhi
3	Charlie	Bangalore
4	David	Mumbai

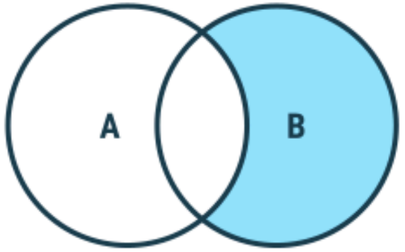
customers

order_id	customer_id	amount
101	1	500
102	1	900
103	2	300
104	5	700

orders



Left Exclusive Join

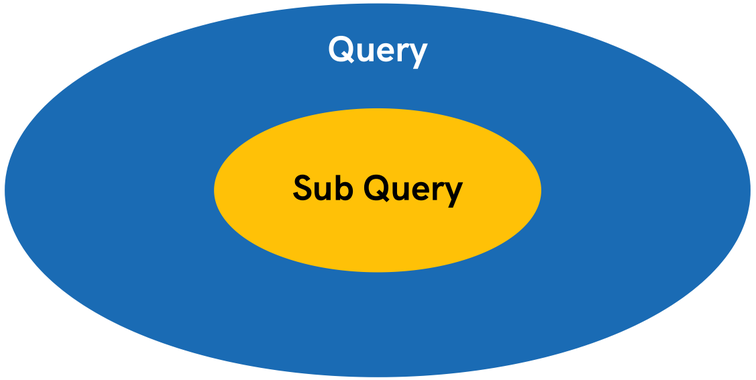


Right Exclusive Join

```sql
-- left exclusive
SELECT *
FROM customers as c
LEFT JOIN orders as o
ON c.customer_id = o.customer_id
WHERE o.customer_id IS NULL;
```

```sql
-- right exclusive
SELECT *
FROM customers as c
RIGHT JOIN orders as o
ON c.customer_id = o.customer_id
WHERE c.customer_id IS NULL;
```

Sub-Queries



A Subquery or Inner query or a Nested query is a query within another SQL query. It involves 2 select statements.

Syntax

```
SELECT column(s)
FROM table_name
WHERE col_name operator
( subquery );
```

Sub-Queries

With WHERE

```sql
SELECT *
FROM orders
WHERE amount > (
    SELECT AVG(amount)
    FROM orders
);
```

Sub-Queries

With SELECT

```sql
SELECT name,
    (   SELECT COUNT(*)
        FROM orders o
        WHERE o.customer_id = c.customer_id
    ) as order_count
FROM customers c;
```

Sub-Queries

With FROM

```sql
SELECT
    summary.customer_id,
    summary.avg_amount
FROM
    (
        SELECT
            customer_id,
            AVG(amount) AS avg_amount
        FROM orders
        GROUP BY customer_id
    ) AS summary;
```

Views in SQL

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

Syntax

CREATE VIEW *view1 AS*
SELECT *col1, col2* **FROM** *table_name;*

*A view always shows up-to-date data.
The database engine recreates the view,
every time a user queries it.

Views in SQL

- No data is stored physically (unless it's a materialized view in some DBs).
- Can include columns from one or more tables.
- Can be used in SELECT, JOIN, or even WHERE clauses like a normal table.
- Helps with security by exposing only certain columns to users.

Index in SQL

indexes are special database objects that make data retrieval faster.

Syntax (single col & multi-col)

CREATE INDEX *idx_name* ON *table(col);*

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

SHOW INDEX FROM *table;*

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 procedure_name (parameters)
BEGIN
    -- SQL statements
END;
```



```
DELIMITER $$

CREATE PROCEDURE check_balance(IN acc_id INT, OUT bal DECIMAL(10, 2))
BEGIN
    SELECT balance INTO bal
    FROM bank_accounts as b
    WHERE b.account_id = acc_id;
END $$

DELIMITER ;

CALL check_balance(2, @balance);
SELECT @balance;
```

Stored Procedures

Syntax (Call)

CALL *procedure_name (arguments);*

Syntax (Drop)

DROP PROCEDURE **IF EXISTS** *procedure_name;*