**🔹 What are Set Operators?**

Set operators are used to **combine results from multiple SELECT queries**.

**1. UNION**

* Combines results of 2 queries.
* Removes duplicates (like DISTINCT).

SELECT customer\_name FROM customers

UNION

SELECT customer\_name FROM vip\_customers;

👉 Output = distinct names from both tables.

**2. UNION ALL**

* Same as UNION, but keeps duplicates.

SELECT customer\_name FROM customers

UNION ALL

SELECT customer\_name FROM vip\_customers;

👉 Output = all names, duplicates included.

**3. INTERSECT**

* Returns **common rows** between two queries.

SELECT customer\_name FROM customers

INTERSECT

SELECT customer\_name FROM vip\_customers;

👉 Output = names present in both tables.  
⚠️ Not supported in MySQL directly. You simulate using INNER JOIN.

**4. EXCEPT (also called MINUS in Oracle)**

* Returns rows from **first query** that are **not in second query**.

SELECT customer\_name FROM customers

EXCEPT

SELECT customer\_name FROM vip\_customers;

👉 Output = customers who are not VIP.  
⚠️ Not supported in MySQL directly. You simulate using LEFT JOIN + WHERE NULL.

**✅ MySQL Compatibility**

* MySQL supports: **UNION** and **UNION ALL**.
* MySQL doesn’t support: **INTERSECT** and **EXCEPT** directly → we use **joins** or **NOT EXISTS** instead.

1. Note : To UNION, both SELECT queries must return the same number of columns with compatible data types.
2. That’s an excellent **why-question** 👍 and exactly what interviewers love asking.  
   You’re right — we *can* often solve things with JOINs, but **UNION is not the same thing** as JOIN.

**🔄 JOIN vs UNION (key difference)**

**1. JOIN → Combine columns (side by side)**

* Used when you want to **bring related data together from multiple tables**.
* Works on relationships (ON condition).
* Output = more **columns**.
* Result: customer\_name | account\_type  
  (Side by side info).

**2. UNION → Combine rows (stack vertically)**

* Used when you want to **combine results of two different SELECT queries**.
* No relationship needed, only same number of columns with compatible datatypes.
* Output = more **rows**.
* 👉 Example:
* SELECT c.customer\_name FROM customers
* UNION
* SELECT c.customer\_name FROM accounts a
* JOIN customers c ON a.customer\_id = c.customer\_id;

**⚡ When do we actually need UNION?**

1. **Different data sources** → e.g.,
   * customers table stores retail customers.
   * vendors table stores suppliers.  
     You want a single list of **all names (customers + vendors)**.  
     That’s not a JOIN, that’s UNION.
2. **Different filtering conditions** → e.g.,
   * Get customers from *Delhi*
   * Get customers whose balance > 50k  
     Combine both into one list.
3. SELECT customer\_name, city FROM customers WHERE city = 'Delhi'
4. UNION
5. SELECT customer\_name, city FROM customers WHERE balance > 50000;
6. **History / Archive tables** → e.g.,
   * accounts\_2024
   * accounts\_2025  
     Combine both years into one query using UNION ALL.

✅ Rule of Thumb:

* Use **JOIN** → when you need **more info about the same record** (combine columns).
* Use **UNION** → when you need to **stack results together** (combine rows).

👉 In **UNION**, both SELECT statements must return the **same number of columns** and with compatible data types.

erfect 👌 let’s break down **how CAST works** with some examples in your bank\_db1 context.

**1. Without CAST → Error ⚠️**

SELECT balance FROM accounts

UNION

SELECT customer\_name FROM customers;

❌ This fails because:

* balance = INT (number)
* customer\_name = VARCHAR (string)
* UNION requires **same datatype in both SELECTs**.

**2. With CAST → Works ✅**

SELECT CAST(balance AS CHAR) AS data

FROM accounts

UNION

SELECT customer\_name

FROM customers;

🔎 What happens here:

* CAST(balance AS CHAR) converts numbers → strings.  
  Example: 50000 becomes '50000'.
* Now both queries return VARCHAR, so UNION works.
* Result: one combined list of names and balances.

**3. Another CAST Example (date)**

If you had a created\_at column (DATE type), but you want to UNION it with names:

SELECT CAST(created\_at AS CHAR) FROM accounts

UNION

SELECT customer\_name FROM customers;

➡️ It converts date like 2025-09-02 → '2025-09-02' (string).

👉 **Rule of thumb**:  
CAST is like telling SQL:  
“Treat this value as type X so it matches the other side.”

Would you like me to **show you a step-by-step demo** with a small fake dataset (customers + accounts) so you see exactly how CAST changes the values?