order_id	customer_id	subscription_id	purchase date
1	2	3	2017-01-01
2	2	2	2017-01-01
3	3	1	2017-01-01

subscriptions

(a table that describes each type of subscription)

subscription_id	description	price_per_month	length
1	Politics Magazine	5	12 months
2	Fashion Magazine	10	6 months
3	Sports Magazine	7	3 months

customers

(a table with customer names and contact information)

customer_id	customer_name	address
1	John Smith	123 Main St
2	Jane Doe	456 Park Ave
3	Joe Schmo	798 Broadway

Examining the order_id of 2, we see it has a customer_id of 2. So we go to the “customers” table where we see customer_id 2 corresponds to ‘Jane Doe’.

The act of matching shared columns through different tables is called **joining**.

SELECT *

FROM orders

JOIN customers

ON orders.customer_id = customers.customer_id

WHERE description = 'Fashion Magazine';

1. Selects all columns from the combined table.
2. Specifies first table we want to look in, orders.
3. Specifies second table we with to join. So, we want to JOIN orders and customers.
4. Match orders' customer_id column with customers' customer_id column.
5. Selects rows where description value = 'Fashion Magazine'.

SELECT *

FROM table1

LEFT JOIN table2

ON table1.c2 = table2.c2;

Joins table1 and table2 by table1.c2 and table2.c2 columns. However, this keeps unmatched values from column1, regardless of EMPTY or not from column2.

---

order_id	customer_id	subscription_id	purchase_date
1	2	3	2017-01-01
2	2	2	2017-01-01
3	3	1	2017-01-01

order_id, **customer_id**, **subscription_id** are all PRIMARY KEYS for their respective tables. They cannot be NULL, must be unique, and cannot have more than one primary key column.

When **customer_id** and **subscription_id** both appear in 'order' table, they are considered FOREIGN KEYS. When working with multiple FOREIGN KEYS (usually joining two tables by a common primary key), they have more descriptive names. Otherwise they are just called 'id'.

SELECT shirts.shirt_colour, pants.pant_colour

FROM shirts

CROSS JOIN pants;

Displays shirt colours, pant colours columns next to each other. From shirts table, we CROSS JOIN pants table where each each shirt_color is matched to each pant_color, basically creating a table of all the possible shirt/pant colour combinations.

SELECT *

FROM table1

UNION

SELECT *

FROM table2

Stacks one table's data onto another, literally. MUST HAVE SAME NUMBER OF COLUMNS. MUST HAVE SAME COLUMN DATA TYPES.

WITH <new_table> AS (

    SELECT …

    …

    …

)

SELECT *

FROM <new_table>

JOIN customers

    ON customers.id = customers.name;

WITH allows us to perform a separate query (such as aggregating customer's subs) stored in a temporary table. We can later reference the temporary table.

```
WITH previous_query AS (
   SELECT customer_id,
      COUNT(subscription_id) AS 'subscriptions'
   FROM orders
   GROUP BY customer_id
)
SELECT customers.customer_name,
   previous_query.subscriptions
FROM previous_query
JOIN customers
  ON previous_query.customer_id =
customers.customer_id;
```

`JOIN` will combine rows from different tables if the join condition is true.

`LEFT JOIN` will return every row in the *left* table, and if the join condition is not met, `NULL` values are used to fill in the columns from the *right* table.

Primary key is a column that serves a unique identifier for the rows in the table.

Foreign key is a column that contains the primary key to another table.

`CROSS JOIN` lets us combine all rows of one table with all rows of another table.

`UNION` stacks one dataset on top of another.

`WITH` allows us to define one or more temporary tables that can be used in the final query.