1. JOINs: Used to Collect Data from Two or More Tables

- Cartesian Product: when there is no relation between two tables:
Each row in the first table is paired with all rows in the second table
Formed when the join condition is omitted
To avoid this, always include a valid JOIN condition

```
SELECT
      last_name,
      name AS department_name
FROM
      employees,
      departments;
```

- Inner Join: produces a set of records that match in both tables

```
SELECT
      e.first_name,
      d.name
FROM employees AS e
INNER JOIN departments --or just JOIN
ON e.department_id = d.department_id
```

- Left Join: matches every entry in left table regardless of match in the right

```
SELECT
      e.first_name,
      d.name
FROM employees AS e
LEFT JOIN departments
ON e.department_id = d.department_id
```

- Right Join: matches every entry in right table regardless of match in the left

```
SELECT
      e.first_name,
      d.name
FROM employees AS e
LEFT JOIN departments
ON e.department_id = d.department_id
```

- Full Join: returns all records in both tables regardless of any match

```
SELECT
      e.first_name,
      d.name
FROM employees AS e
FULL JOIN departments
ON e.department_id = d.department_id
```

- Cross Join: produces a set of associated rows of two tables
Multiplication of each row in the first table with each row in the second one
The result is a cartesian product where there is no condition in WHERE clause

```
SELECT
      d.department_id,
      d.name,
      e.employee_id
      e.first_name
FROM
      departments AS d
CROSS JOIN employees as e;
```

2. Subqueries: query manipulation on multiple level
   - Subqueries: SQL query inside a larger one
   - Can be nested in SELECT, INSERT, UPDATE, DELETE
   - Usually added within a WHERE clause

```
SELECT
      employee_id AS id,
      first_name,
      department_ud
FROM
      employees
WHERE
      department_id = 1;
```

```
SELECT
      COUNT(e.employee_id) AS count
FROM
      employees as e
WHERE e.salary >
(
      SELECT
            AVG(salary) AS "average_salary"
      FROM
      employees
);
```

3. Indices: Speed Retrieval of Rows
   - Special lookup tables that speed retrieval of rows
   - Usually implemented as B-trees
   - Speed up SELECt queries and WHERE queries
   - Slows down data input
   - Should be used for big tables only(e.g.: 50 000rows)
   - Should not be used on columns that contain a high number of NULL values

```
CREATE INDEX index_name_index
ON table_name(first_column, second_column);
```

clustered and non-clustered