4.Data Selection Techniques in MS-SQL Server.

4.1 Basic SQL Server select statement

```
SELECT
    select_list
FROM
    schema_name.table_name;
```

```
SELECT
    first_name,
    last_name
FROM
    sales.customers;
```

4.2 ORDER BY

Introduction to the SQL Server ORDER BY clause

When you use the SELECT statement to query data from a table, the order of rows in the result set is not guaranteed. It means that SQL Server can return a result set with an unspecified order of rows.

A) Sort a result set by one column in ascending order :

```
SELECT
    select_list
FROM
    table_name
ORDER BY
    [column_name | expression] [ASC | DESC ]
```

```
SELECT
    first_name,
    last_name
FROM
    sales.customers
ORDER BY
    first_name;
```

B) Sort a result set by one column in descending order

```
SELECT
firstname,
lastname
FROM
sales.customers
ORDER BY
first_name DESC;
```

C) Sort a result set by multiple columns

```
SELECT
    city,
    first_name,
    last_name
FROM
    sales.customers
ORDER BY
    city,
    first_name;
```

D) Sort a result set by multiple columns and different orders

```
SELECT
    city,
    first_name,
    last_name
FROM
    sales.customers
ORDER BY
    city DESC,
    first_name ASC;
```

F) Sort a result set by an expression

```
SELECT
    first_name,
    last_name
FROM
    sales.customers
ORDER BY
    LEN(first_name) DESC;
```

4.3 SELECT TOP

Introduction to SQL Server **SELECT TOP**:

The SELECT TOP clause allows you to limit the number of rows

or percentage of rows returned in a query result set.

```
SELECT TOP (expression) [PERCENT]
    [WITH TIES]
FROM
    table_name
ORDER BY
    column_name;
```

A) Using TOP with a constant value

```
SELECT TOP 10
    product_name,
    list_price
FROM
    production.products
ORDER BY
    list_price DESC;
```

B) Using TOP to return a percentage of rows

```
SELECT TOP 1 PERCENT
    product_name,
    list_price
FROM
    production.products
ORDER BY
    list_price DESC;
```

C) Using TOP WITH TIES to include rows that match the values in the last row

```
SELECT TOP 3 WITH TIES
    product_name,
    list_price
FROM
    production.products
ORDER BY
    list_price DESC;
```

4.4 DISTINCT

Introduction to SQL Server SELECT DISTINCT clause.

```
SELECT DISTINCT
    column_name
FROM
    table_name;
```

A) DISTINCT one column example.

```
SELECT
    city
FROM
    sales.customers
ORDER BY
    city;
```

B) DISTINCT multiple columns example.

```
SELECT DISTINCT
    city,
    state
FROM
    sales.customers
```

4.5 WHERE clause

Introduction to SQL Server WHERE clause

```
SELECT
    select_list
FROM
    table_name
WHERE
    search_condition;
```

A) Finding rows by using a simple equality.

```
SELECT
    product_id,
    product_name,
    category_id,
    model_year,
    list_price
FROM
    production.products
WHERE
    category_id = 1
ORDER BY
    list_price DESC;
```

B) Finding rows that meet two conditions.

```sql
SELECT
    product_id,
    product_name,
    category_id,
    model_year,
    list_price
FROM
    production.products
WHERE
    category_id = 1 AND model_year = 2018
ORDER BY
    list_price DESC;
```

C) Finding rows by using a comparison operator.

```sql
SELECT
    product_id,
    product_name,
    category_id,
    model_year,
    list_price
FROM
    production.products
WHERE
    list_price > 300 AND model_year = 2018
ORDER BY
    list_price DESC;
```

D) Finding rows that meet any of two conditions.

```sql
SELECT
    product_id,
    product_name,
    category_id,
    model_year,
    list_price
FROM
    production.products
WHERE
    list_price > 3000 OR model_year = 2018
ORDER BY
    list_price DESC;
```

E) Finding rows with the value between two values.

The following statement finds the products whose list prices are between 1,899 and 1,999.99:

```
SELECT
    product_id,
    product_name,
    category_id,
    model_year,
    list_price
FROM
    production.products
WHERE
    list_price BETWEEN 1899.00 AND 1999.99
ORDER BY
    list_price DESC;
```

F) Finding rows that have a value in a list of values.

The following example uses the IN operator to find products whose list price is 299.99 or 466.99 or 489.99.

```
SELECT
    product_id,
    product_name,
    category_id,
    model_year,
    list_price
FROM
    production.products
WHERE
    list_price IN (299.99, 369.99, 489.99)
ORDER BY
    list_price DESC;
```

4.6 NULL

```sql
SELECT
    customer_id,
    first_name,
    last_name,
    phone
FROM
    sales.customers
WHERE
    phone = NULL
ORDER BY
    first_name,
    last_name;
```

```sql
SELECT
    customer_id,
    first_name,
    last_name,
    phone
FROM
    sales.customers
WHERE
    phone IS NULL
ORDER BY
    first_name,
    last_name;
```

```sql
SELECT
    customer_id,
    first_name,
    last_name,
    phone
FROM
    sales.customers
WHERE
    phone IS NOT NULL
ORDER BY
    first_name,
    last_name;
```

4.7 LIKE

The **%** (percent) wildcard example.The following example finds the customers whose last name starts with the letter z.

```
SELECT
    customer_id,
    first_name,
    last_name
FROM
    sales.customers
WHERE
    last_name LIKE 'z%'
ORDER BY
    first_name;
```

The following example returns the customers whose last name ends with the string e.g:

```
SELECT
    customer_id,
    first_name,
    last_name
FROM
    sales.customers
WHERE
    last_name LIKE '%er'
ORDER BY
    first_name;
```

The following statement retrieves the customers whose last name starts with the letter t and ends with the letter s:

```
SELECT
    customer_id,
    first_name,
    last_name
FROM
    sales.customers
WHERE
    last_name LIKE 't%s'
ORDER BY
    first_name;
```

The _ (underscore) wild card example: The underscore represents a single character. For example, the following statement returns the customers where the second character is the letter u:

```
SELECT
    customer_id,
    first_name,
    last_name
FROM
    sales.customers
WHERE
    last_name LIKE '_u%'
ORDER BY
    first_name;
```

The [list of characters] wildcard example: The square brackets with a list of characters e.g. [ABC] represent a single character that must be one of the characters specified in the list.

For example, the following query returns the customers where the first character in the last name is Y or Z:

```
SELECT
    customer_id,
    first_name,
    last_name
FROM
    sales.customers
WHERE
    last_name LIKE '[YZ]%'
ORDER BY
    last_name;
```

The NOT LIKE operator example.

The following example uses the NOT LIKE operator to find customers where the first character in the first name is not the letter A:

```
SELECT
    customer_id,
    first_name,
    last_name
FROM
    sales.customers
WHERE
    first_name NOT LIKE 'A%'
ORDER BY
    first_name;
```

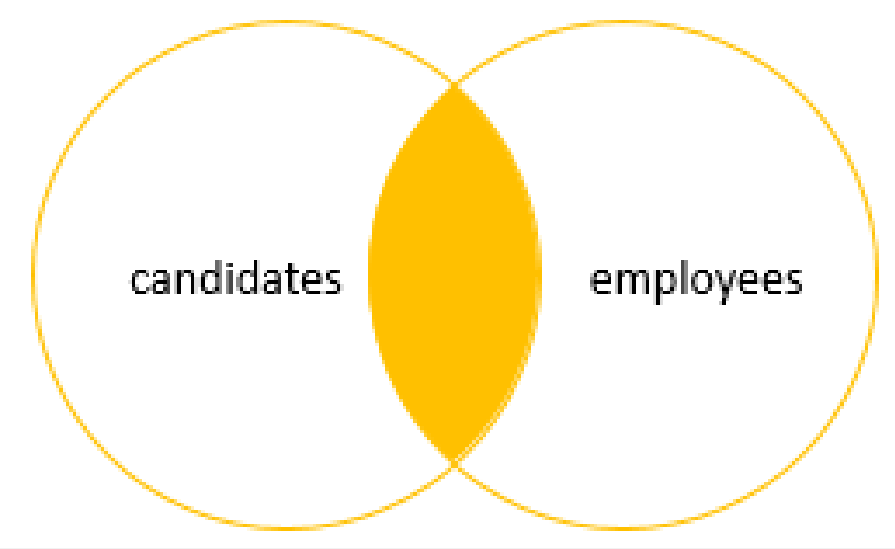
4.8 JOINS

In a relational database, data is distributed in multiple logical tables. To get a complete meaningful set of data, you need to query data from these tables by using joins.

SQL Server supports many kinds of joins including inner join, left join, right join, full outer join, and cross join.

A) SQL Server Inner Join

Inner join produces a data set that includes rows from the left table which have matching rows from the right table.



Setting up sample tables

```
CREATE SCHEMA hr;
GO
```

```
CREATE TABLE hr.candidates(
    id INT PRIMARY KEY IDENTITY,
    fullname VARCHAR(100) NOT NULL
);

CREATE TABLE hr.employees(
    id INT PRIMARY KEY IDENTITY,
    fullname VARCHAR(100) NOT NULL
);
```

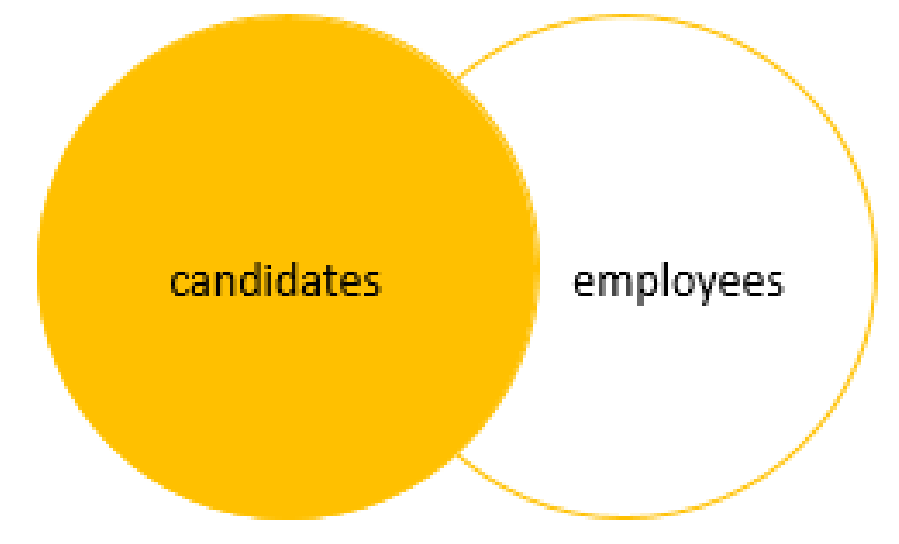
```
INSERT INTO
    hr.candidates(fullname)
VALUES
    ('John Doe'),
    ('Lily Bush'),
    ('Peter Drucker'),
    ('Jane Doe');


INSERT INTO
    hr.employees(fullname)
VALUES
    ('John Doe'),
    ('Jane Doe'),
    ('Michael Scott'),
    ('Jack Sparrow');
```

```sql
SELECT
    c.id candidate_id,
    c.fullname candidate_name,
    e.id employee_id,
    e.fullname employee_name
FROM
    hr.candidates c
    INNER JOIN hr.employees e
        ON e.fullname = c.fullname;
```

B) SQL Server Left Join

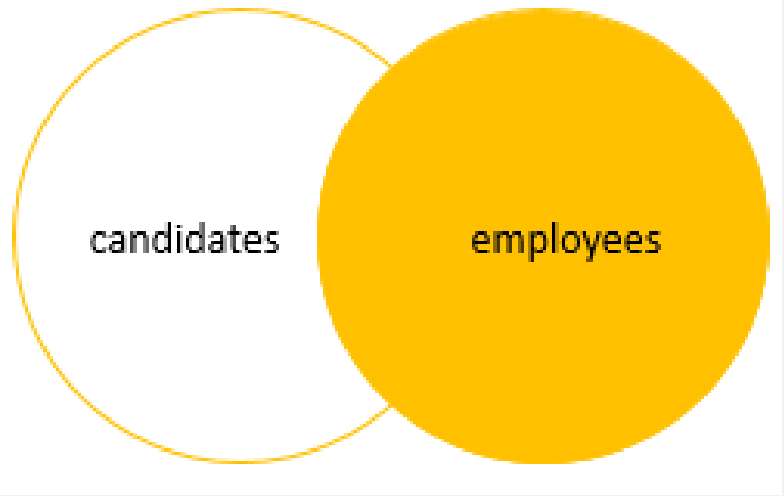
Left join selects data starting from the left table and matching rows in the right table. The left join returns all rows from the left table and the matching rows from the right table. If a row in the left table does not have a matching row in the right table, the columns of the right table will have nulls.



```sql
SELECT
c.id candidate_id,
c.fullname candidate_name,
e.id employee_id,
e.fullname employee_name
FROM
hr.candidates c
LEFT JOIN hr.employees e
ON e.fullname = c.fullname;
```

C) SQL Server Right Join

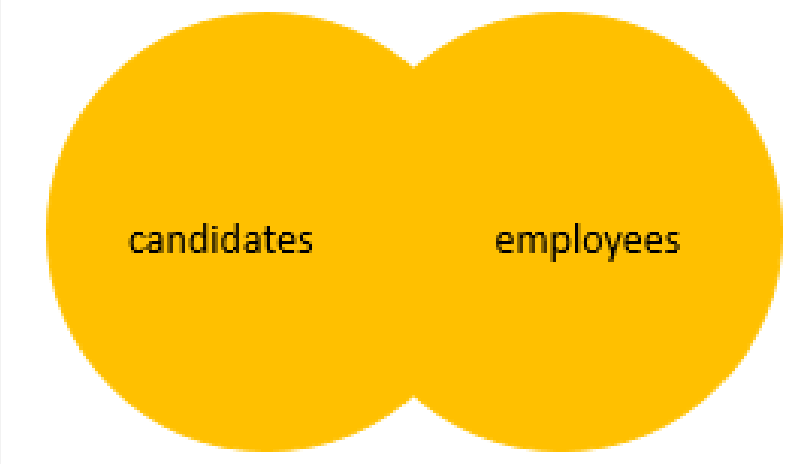
The right join or right outer join selects data starting from the right table. The right join returns a result set that contains all rows from the right table and the matching rows in the left table. If a row in the right table that does not have a matching row in the left table, all columns in the left table will contain nulls.



```sql
SELECT
    c.id candidate_id,
    c.fullname candidate_name,
    e.id employee_id,
    e.fullname employee_name
FROM
    hr.candidates c
    RIGHT JOIN hr.employees e
        ON e.fullname = c.fullname;
```

C) SQL Server Full Join

The full outer join or full join returns a result set that contains all rows from both left and right tables, with the matching rows from both sides where available. In case there is no match, the missing side will have NULL values.



The following shows the syntax of the FULL OUTER JOIN when joining two tables T1 and T2:

```
SELECT
    select_list
FROM
    T1
FULL OUTER JOIN T2 ON join_predicate;
```

The OUTER keyword is optional so you can skip it as shown in the following query:

```
SELECT
    select_list
FROM
    T1
FULL JOIN T2 ON join_predicate;
```

```
CREATE SCHEMA pm;
GO
```

```sql
CREATE TABLE pm.projects(
    id INT PRIMARY KEY IDENTITY,
    title VARCHAR(255) NOT NULL
);

CREATE TABLE pm.members(
    id INT PRIMARY KEY IDENTITY,
    name VARCHAR(120) NOT NULL,
    project_id INT,
    FOREIGN KEY (project_id)
        REFERENCES pm.projects(id)
);
```

```sql
INSERT INTO
    pm.projects(title)
VALUES
    ('New CRM for Project Sales'),
    ('ERP Implementation'),
    ('Develop Mobile Sales Platform');
INSERT INTO
    pm.members(name, project_id)
VALUES
    ('John Doe', 1),
    ('Lily Bush', 1),
    ('Jane Doe', 2),
    ('Jack Daniel', null);


```

```sql
SELECT
    m.name member,
    p.title project
FROM
    pm.members m
    FULL OUTER JOIN pm.projects p
        ON p.id = m.project_id;
```

4.9 GROUP BY

Introduction to SQL Server GROUP BY clause.

The GROUP BY clause allows you to arrange the rows of a query in groups. The groups are determined by the columns that you specify in the GROUP BY clause.

The following illustrates the GROUP BY clause syntax:

```sql
SELECT
    select_list
FROM
    table_name
GROUP BY
    column_name1,
    column_name2 ,...;
```

```sql
SELECT
    customer_id,
    YEAR (order_date) order_year
FROM
    sales.orders
WHERE
    customer_id IN (1, 2)
ORDER BY
    customer_id;
```

SQL Server GROUP BY clause and aggregate functions

```sql
SELECT
    customer_id,
    YEAR (order_date) order_year,
    COUNT (order_id) order_placed
FROM
    sales.orders
WHERE
    customer_id IN (1, 2)
GROUP BY
    customer_id,
    YEAR (order_date)
ORDER BY
    customer_id;
```

Using GROUP BY clause with the COUNT() function example

```
SELECT
    city,
    COUNT (customer_id) customer_count
FROM
    sales.customers
GROUP BY
    city
ORDER BY
    city;
```

The following query uses the SUM() function to get the net value of every order:

```
SELECT
    order_id,
    SUM (
        quantity * list_price * (1 - discount)
    ) net_value
FROM
    sales.order_items
GROUP BY
    order_id;
```

4.10 Having

Introduction to SQL Server HAVING clause:

The HAVING clause is often used with the GROUP BY clause to filter groups based on a specified list of conditions. The following illustrates the HAVING clause syntax:

```
SELECT
    select_list
FROM
    table_name
GROUP BY
    group_list
HAVING
    conditions;
```

The following statement finds the customers who placed at least two orders per year:

```
SELECT
    customer_id,
    YEAR (order_date),
    COUNT (order_id) order_count
FROM
    sales.orders
GROUP BY
    customer_id,
    YEAR (order_date)
HAVING
    COUNT (order_id) >= 2
ORDER BY
    customer_id;
```

The following statement finds the sales orders whose net values are greater than 20,000:

```
SELECT
    order_id,
    SUM (
        quantity * list_price * (1 - discount)
    ) net_value
FROM
    sales.order_items
GROUP BY
    order_id
HAVING
    SUM (
        quantity * list_price * (1 - discount)
    ) > 20000
ORDER BY
    net_value;
```

4.11 Subquery

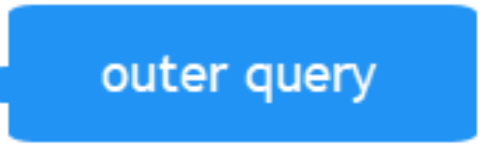
Introduction to SQL Server subquery.

A subquery is a query nested inside another statement such as SELECT, INSERT, UPDATE, or DELETE.Let's see the following example.

The following statement shows how to use a subquery in the WHERE clause of a SELECT statement to find the sales orders of the customers who locate in New York:

```
SELECT
    order_id,
    order_date,
    customer_id
FROM
    sales.orders
WHERE
    customer_id IN (
        SELECT
            customer_id
        FROM
            sales.customers
        WHERE
            city = 'New York'
    )
ORDER BY
    order_date DESC;
```

```sql
SELECT
    order_id,
    order_date,
    customer_id
FROM
    sales.orders
WHERE
    customer_id IN (
        SELECT
            customer_id
        FROM
            sales.customers
        WHERE
            city = 'New York'
    )
ORDER BY
    order_date DESC;
```



SQL Server subquery types:

A.In place of an expression

B.With IN or NOT IN

C.With ANY or ALL

D.With EXISTS or NOT EXISTS

E.In UPDATE, DELETE, or INSERT statement.

A.SQL Server subquery is used in place of an expression

```
SELECT
    order_id,
    order_date,
    (
        SELECT
            MAX (list_price)
        FROM
            sales.order_items i
        WHERE
            i.order_id = o.order_id
    ) AS max_list_price
FROM
    sales.orders o
order by order_date desc;
```

B. SQL Server subquery is used with IN operator. A subquery which is used with the IN operator returns a set of zero or more values. After the subquery returns values, the outer query makes use of them.

```
SELECT
    product_id,
    product_name
FROM
    production.products
WHERE
    category_id IN (
        SELECT
            category_id
        FROM
            production.categories
        WHERE
            category_name = 'Mountain Bikes'
        OR category_name = 'Road Bikes'
    );
```

C. SQL Server subquery is used with ANY operator. The following query finds the products whose list price is greater than or equal to the maximum list price returned by the subquery:

```
SELECT
    product_name,
    list_price
FROM
    production.products
WHERE
    list_price >= ALL (
        SELECT
            AVG (list_price)
        FROM
            production.products
        GROUP BY
            brand_id
    )
```

D. SQL Server subquery is used with EXISTS or NOT EXISTS. The EXISTS operator returns TRUE if the subquery return results; otherwise it returns FALSE.

On the other hand, the NOT EXISTS is opposite to the EXISTS operator.

The following query finds the customers who bought products in 2017:

```
SELECT
    customer_id,
    first_name,
    last_name,
    city
FROM
    sales.customers c
WHERE
    EXISTS (
        SELECT
            customer_id
        FROM
            sales.orders o
        WHERE
            o.customer_id = c.customer_id
        AND YEAR (order_date) = 2017
    )
ORDER BY
    first_name,
    last_name;
```

4.12 UNION

Introduction to SQL Server UNION: SQL Server UNION is one of the set operations that allows you to combine results of two SELECT statements into a single result set which includes all the rows that belongs to the SELECT statements in the union.

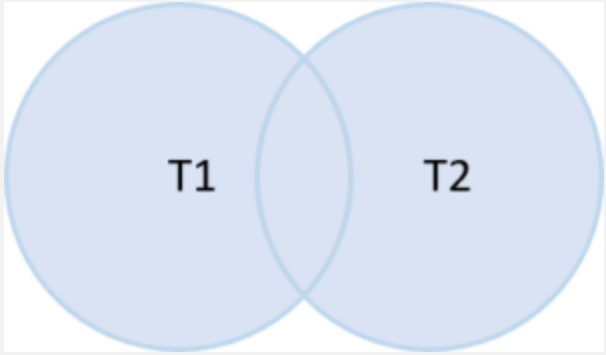
The following illustrates the syntax of the SQL Server UNION:

```
query_1
UNION
query_2
```

The following are requirements for the queries in the syntax above:

- The number and the order of the columns must be the same in both queries.
- The data types of the corresponding columns must be the same or compatible.

The following Venn diagram illustrates how the result set of the T1 table unions with the result set of the T2 table:



```
1  SELECT
2      first_name,
3      last_name
4  FROM
5      sales.staffs
6  UNION
7  SELECT
8      first_name,
9      last_name
1  FROM
0      sales.customers;
11
```

UNION vs. JOIN

The join such as INNER JOIN or LEFT JOIN combines **columns** from two tables while the UNION combines **rows** from two queries.

In other words, join appends the result sets horizontally while union appends result set vertically.

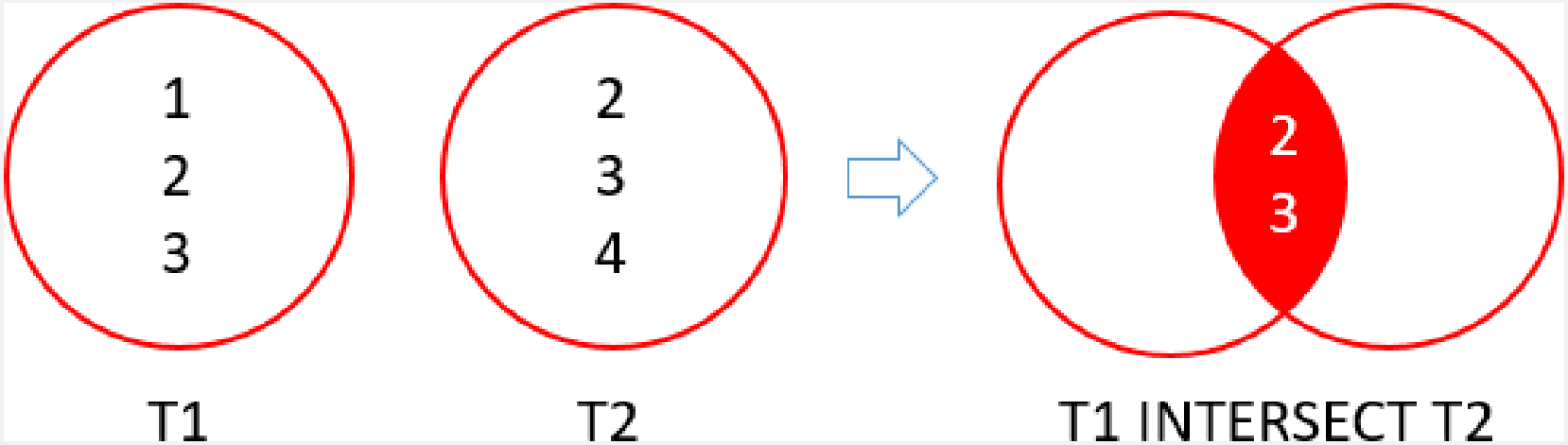
To include the duplicate row, you use the UNION ALL as shown in the following query:

```
SELECT
    first_name,
    last_name
FROM
    sales.staffs
UNION ALL
SELECT
    first_name,
    last_name
FROM
    sales.customers;
```

4.13 INTERSECT

The SQL Server INTERSECT combines result sets of two or more queries and returns distinct rows that are output by both queries.

```
query_1
INTERSECT
query_2
```

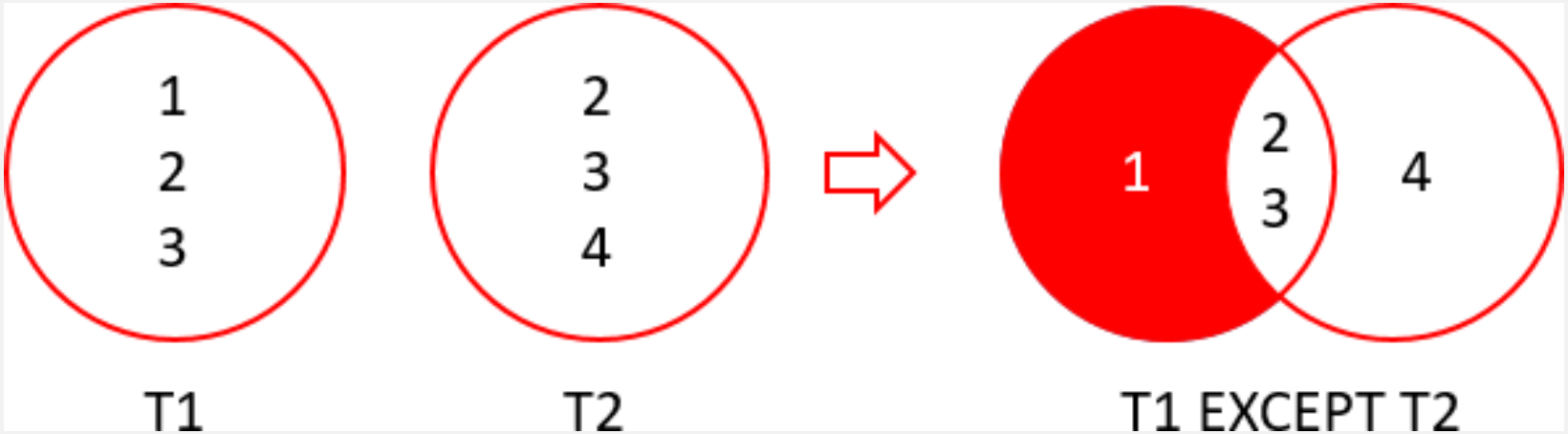


```sql
SELECT
    city
FROM
    sales.customers
INTERSECT
SELECT
    city
FROM
    sales.stores
ORDER BY
    city;
```

4.14 EXCEPT

The SQL Server EXCEPT compares the result sets of two queries and returns the distinct rows from the first query that are not output by the second query. In other words, the EXCEPT subtracts the result set of a query from another.

```
query_1
EXCEPT
query_2
```



```sql
SELECT
    product_id
FROM
    production.products
EXCEPT
SELECT
    product_id
FROM
    sales.order_items;
```