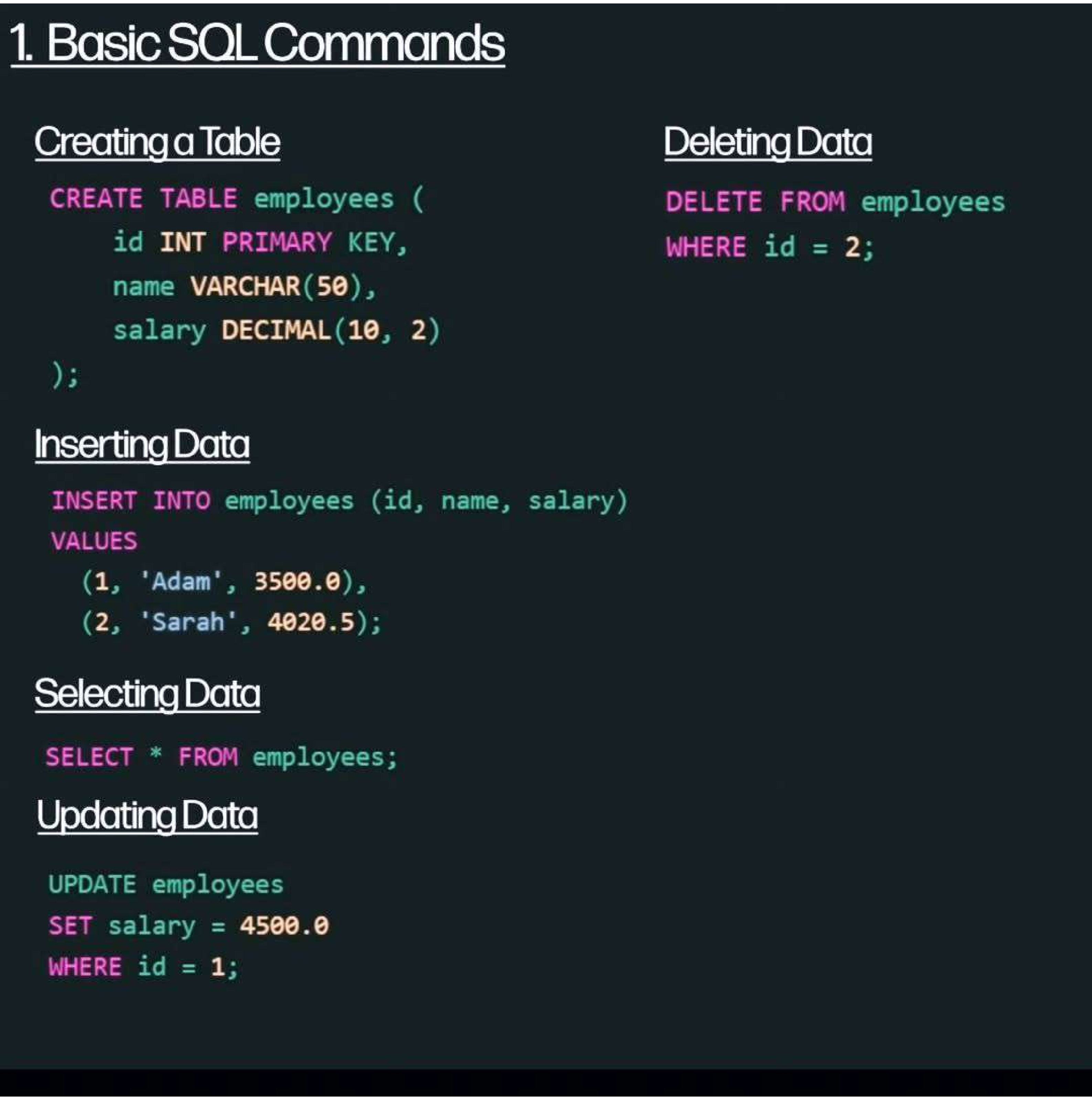
SQL Handbook

SQL



1. Basic SQL Commands

Creating a Table

```
CREATE TABLE employees (
     id INT PRIMARY KEY,
     name VARCHAR(50),
     salary DECIMAL(10, 2)
);
```

Inserting Data

```
INSERT INTO employees (id, name, salary)
VALUES
  (1, 'Adam', 3500.0),
  (2, 'Sarah', 4020.5);
```

Selecting Data

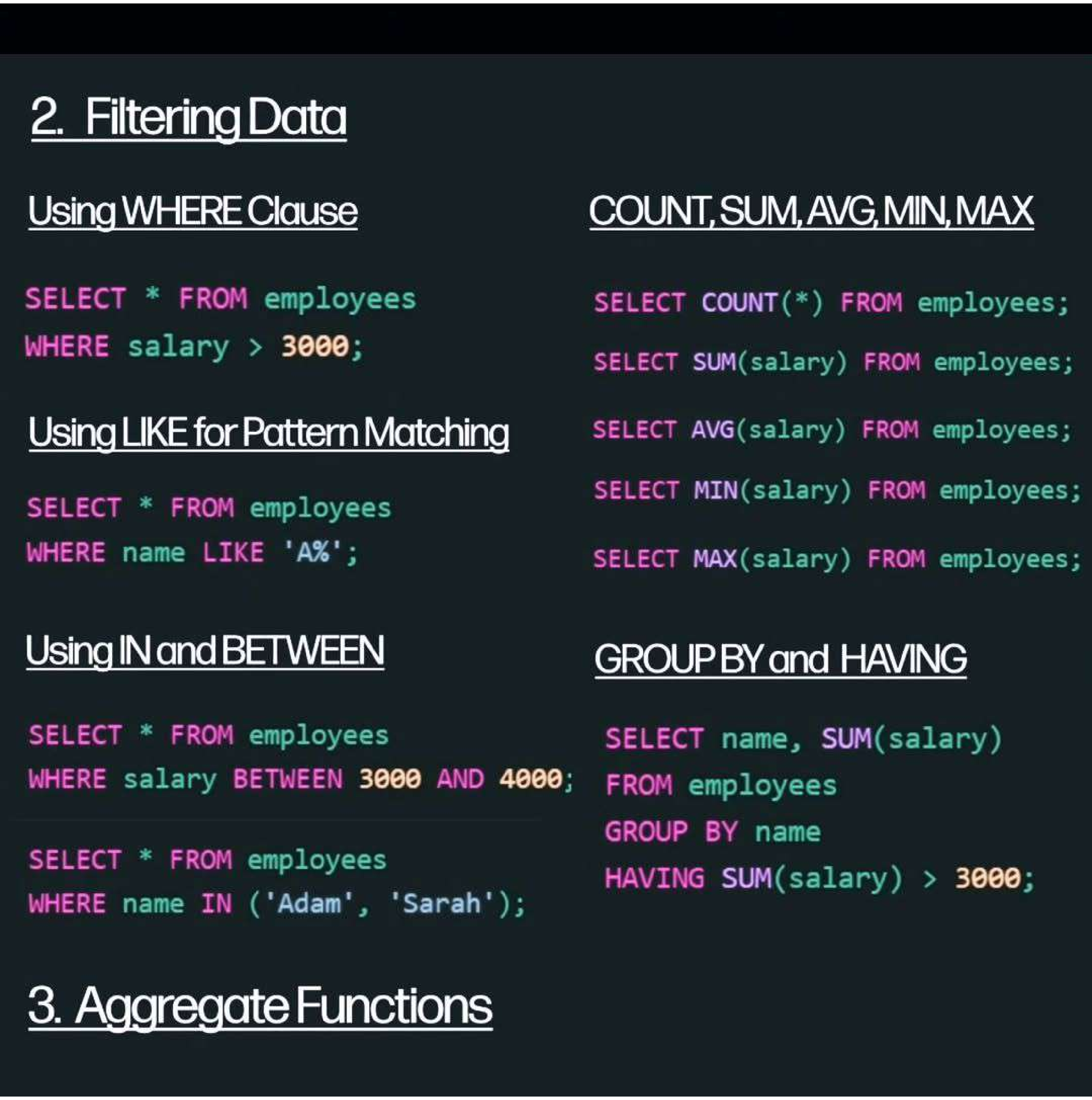
```
SELECT * FROM employees;
```

Updating Data

```
UPDATE employees
SET salary = 4500.0
WHERE id = 1;
```

Deleting Data

```
DELETE FROM employees
WHERE id = 2;
```



2. Filtering Data

Using WHERE Clause

```sql
SELECT * FROM employees
WHERE salary > 3000;
```

Using LIKE for Pattern Matching

```sql
SELECT * FROM employees
WHERE name LIKE 'A%';
```

Using IN and BETWEEN

```sql
SELECT * FROM employees
WHERE salary BETWEEN 3000 AND 4000;
```

```sql
SELECT * FROM employees
WHERE name IN ('Adam', 'Sarah');
```

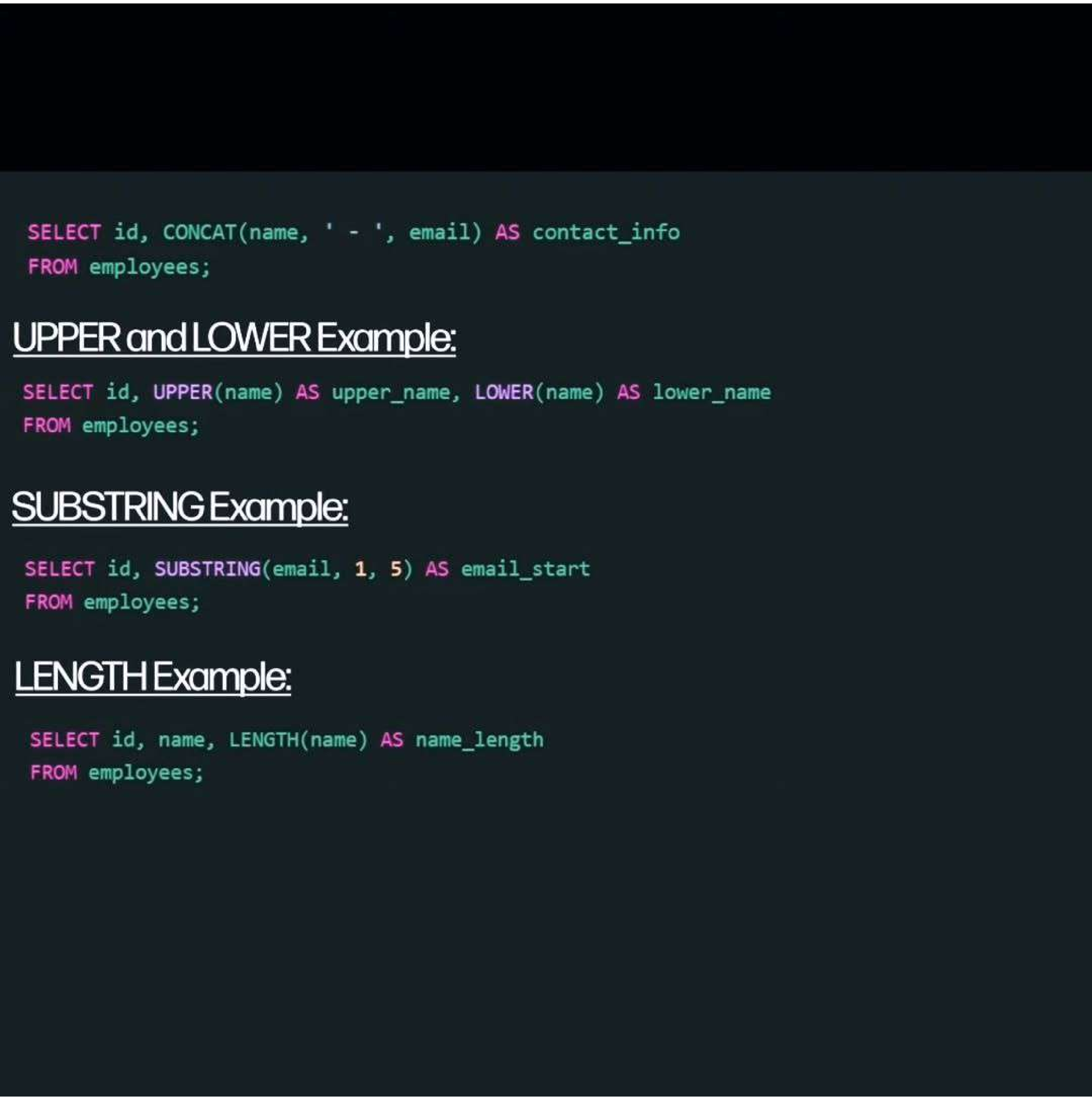
3. Aggregate Functions

COUNT, SUM, AVG, MIN, MAX

```sql
SELECT COUNT(*) FROM employees;

SELECT SUM(salary) FROM employees;

SELECT AVG(salary) FROM employees;

SELECT MIN(salary) FROM employees;

SELECT MAX(salary) FROM employees;
```

GROUP BY and HAVING

```sql
SELECT name, SUM(salary)
FROM employees
GROUP BY name
HAVING SUM(salary) > 3000;
```



```sql
SELECT id, CONCAT(name, ' - ', email) AS contact_info
FROM employees;
```

UPPER and LOWER Example:

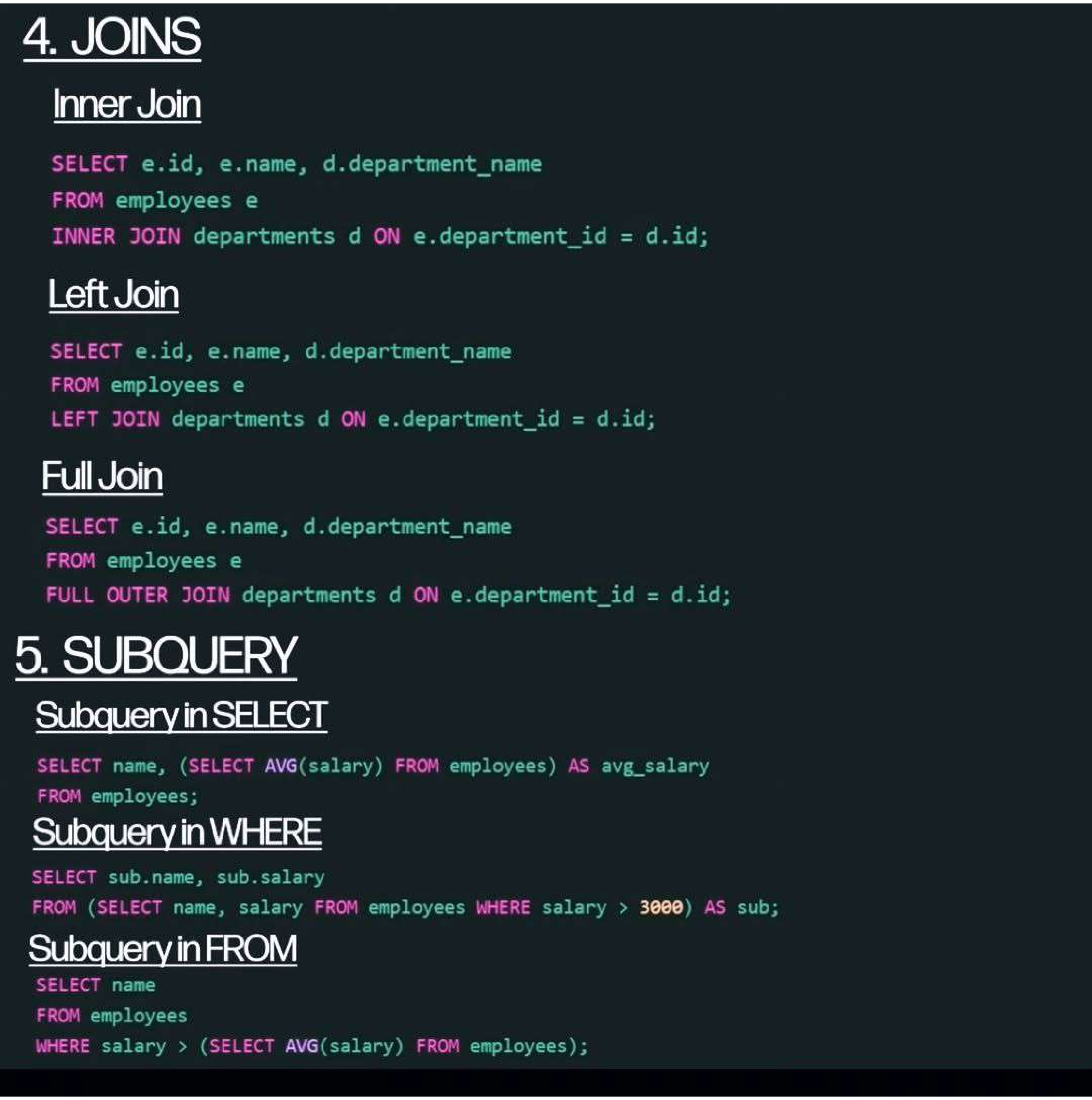
```sql
SELECT id, UPPER(name) AS upper_name, LOWER(name) AS lower_name
FROM employees;
```

SUBSTRING Example:

```sql
SELECT id, SUBSTRING(email, 1, 5) AS email_start
FROM employees;
```

LENGTH Example:

```sql
SELECT id, name, LENGTH(name) AS name_length
FROM employees;
```



4. JOINS

Inner Join

```sql
SELECT e.id, e.name, d.department_name
FROM employees e
INNER JOIN departments d ON e.department_id = d.id;
```

Left Join

```sql
SELECT e.id, e.name, d.department_name
FROM employees e
LEFT JOIN departments d ON e.department_id = d.id;
```

Full Join

```sql
SELECT e.id, e.name, d.department_name
FROM employees e
FULL OUTER JOIN departments d ON e.department_id = d.id;
```

5. SUBQUERY

Subquery in SELECT

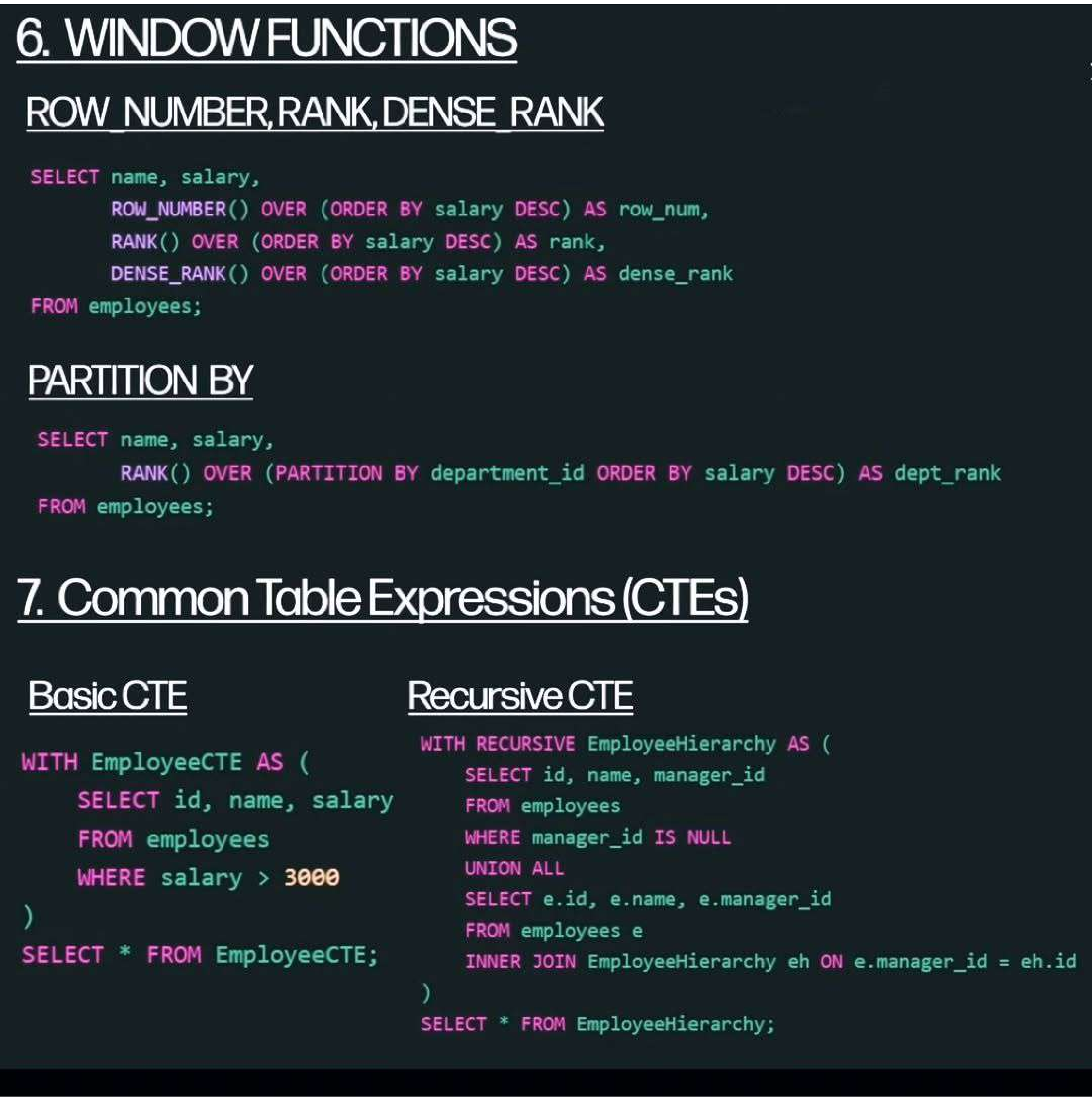
```sql
SELECT name, (SELECT AVG(salary) FROM employees) AS avg_salary
FROM employees;
```

Subquery in WHERE

```sql
SELECT sub.name, sub.salary
FROM (SELECT name, salary FROM employees WHERE salary > 3000) AS sub;
```

Subquery in FROM

```sql
SELECT name
FROM employees
WHERE salary > (SELECT AVG(salary) FROM employees);
```



6. WINDOW FUNCTIONS

ROW_NUMBER, RANK, DENSE_RANK

```sql
SELECT name, salary,
       ROW_NUMBER() OVER (ORDER BY salary DESC) AS row_num,
       RANK() OVER (ORDER BY salary DESC) AS rank,
       DENSE_RANK() OVER (ORDER BY salary DESC) AS dense_rank
FROM employees;
```

PARTITION BY

```sql
SELECT name, salary,
       RANK() OVER (PARTITION BY department_id ORDER BY salary DESC) AS dept_rank
FROM employees;
```

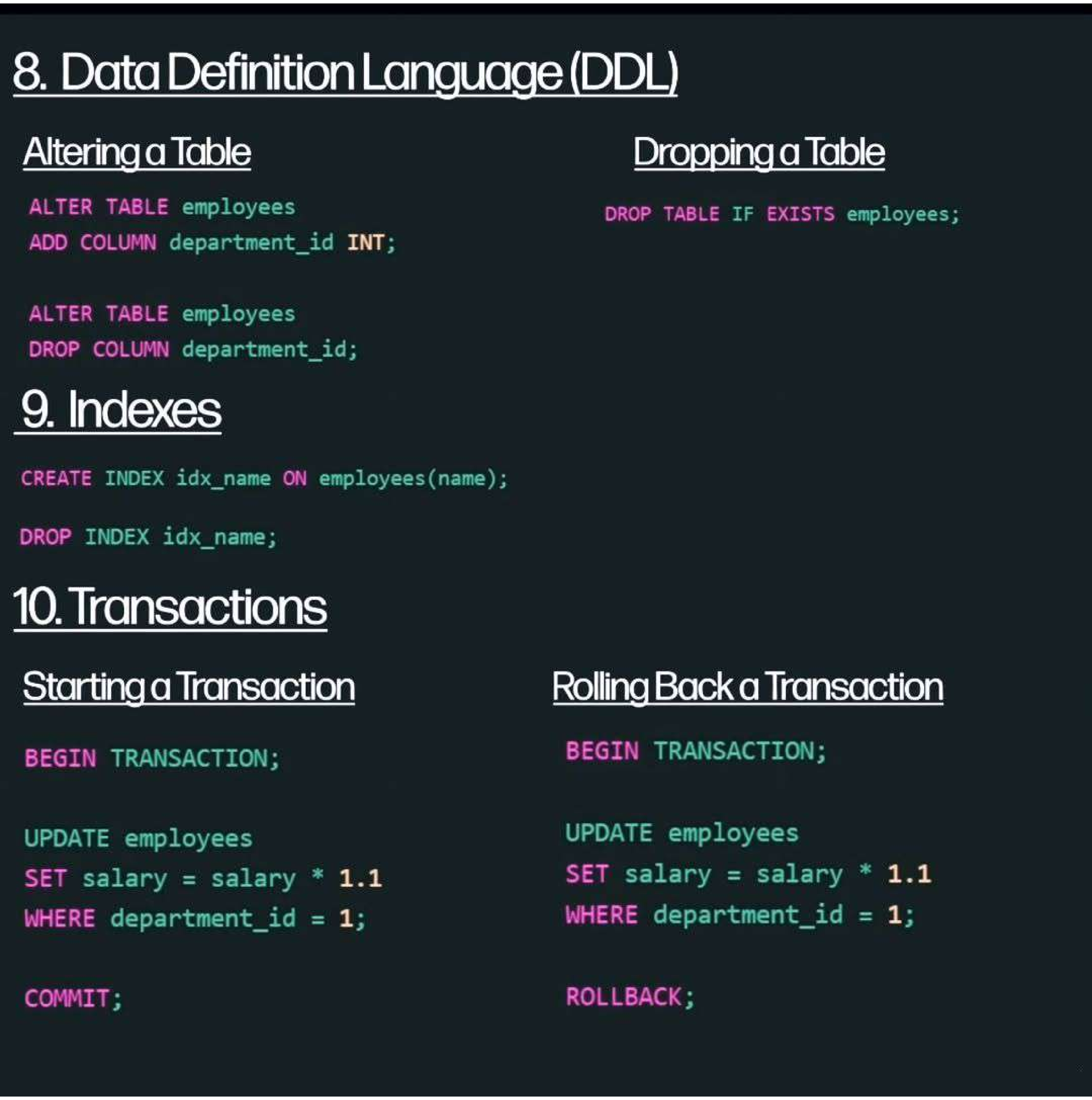
7. Common Table Expressions (CTEs)

Basic CTE

```sql
WITH EmployeeCTE AS (
    SELECT id, name, salary
    FROM employees
    WHERE salary > 3000
)
SELECT * FROM EmployeeCTE;
```

Recursive CTE

```sql
WITH RECURSIVE EmployeeHierarchy AS (
    SELECT id, name, manager_id
    FROM employees
    WHERE manager_id IS NULL
    UNION ALL
    SELECT e.id, e.name, e.manager_id
    FROM employees e
    INNER JOIN EmployeeHierarchy eh ON e.manager_id = eh.id
)
SELECT * FROM EmployeeHierarchy;
```



8. Data Definition Language (DDL)

Altering a Table

```sql
ALTER TABLE employees
ADD COLUMN department_id INT;

ALTER TABLE employees
DROP COLUMN department_id;
```

Dropping a Table

```sql
DROP TABLE IF EXISTS employees;
```

9. Indexes

```sql
CREATE INDEX idx_name ON employees(name);

DROP INDEX idx_name;
```

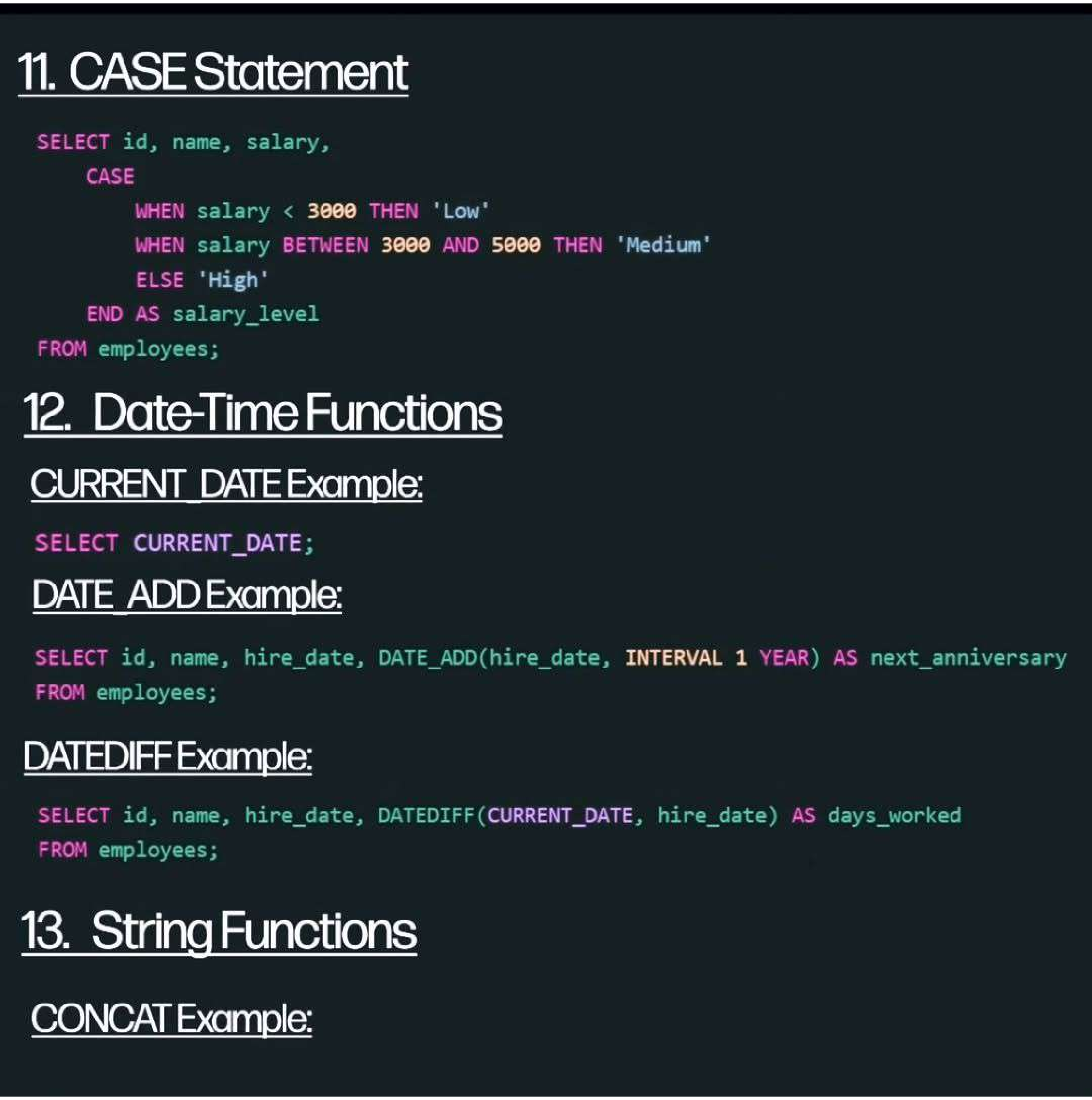
10. Transactions

Starting a Transaction

```sql
BEGIN TRANSACTION;

UPDATE employees
SET salary = salary * 1.1
WHERE department_id = 1;

COMMIT;
```

Rolling Back a Transaction

```sql
BEGIN TRANSACTION;

UPDATE employees
SET salary = salary * 1.1
WHERE department_id = 1;

ROLLBACK;
```



11. CASE Statement

```
SELECT id, name, salary,
    CASE
        WHEN salary < 3000 THEN 'Low'
        WHEN salary BETWEEN 3000 AND 5000 THEN 'Medium'
        ELSE 'High'
    END AS salary_level
FROM employees;
```

12. Date-Time Functions

CURRENT_DATE Example:

```
SELECT CURRENT_DATE;
```

DATE_ADD Example:

```
SELECT id, name, hire_date, DATE_ADD(hire_date, INTERVAL 1 YEAR) AS next_anniversary
FROM employees;
```

DATEDIFF Example:

```
SELECT id, name, hire_date, DATEDIFF(CURRENT_DATE, hire_date) AS days_worked
FROM employees;
```

13. String Functions

CONCAT Example: