Data Manipulation in SQL

CREATE TABLE

```
-- Basic table creation
CREATE TABLE employees (
    id INTEGER PRIMARY KEY,
    name VARCHAR(100) NOT NULL,
    email VARCHAR(150) UNIQUE,
    department VARCHAR(50),
    salary DECIMAL(10,2),
    hire_date DATE
);

-- Table with auto-increment
CREATE TABLE products (
    id INTEGER PRIMARY KEY AUTO_INCREMENT,
    name VARCHAR(100) NOT NULL,
    price DECIMAL(8,2),
    category VARCHAR(50),
    stock INTEGER DEFAULT 0
);
```

INSERT DATA

```
-- Insert single record
INSERT INTO employees (name, email, department, salary, hire_date)
VALUES ('John Smith', 'john@company.com', 'Sales', 50000, '2023-01-15');

-- Insert multiple records
INSERT INTO employees (name, email, department, salary, hire_date)
VALUES
    ('Alice Johnson', 'alice@company.com', 'IT', 65000, '2023-02-01'),
    ('Bob Wilson', 'bob@company.com', 'Sales', 45000, '2023-02-15'),
    ('Carol Brown', 'carol@company.com', 'HR', 55000, '2023-03-01');

-- Insert with some columns
INSERT INTO products (name, price, category)
VALUES ('Laptop', 999.99, 'Electronics');
```

UPDATE DATA

```
-- Update single record
UPDATE employees
SET salary = 52000
WHERE name = 'John Smith';

-- Update multiple columns
UPDATE employees
SET salary = 67000, department = 'Senior IT'
WHERE name = 'Alice Johnson';
```

```
-- Update with conditions
UPDATE products
SET price = price * 0.9
WHERE category = 'Electronics';

-- Update all records (be careful!)
UPDATE employees
SET hire_date = '2023-01-01'
WHERE hire_date IS NULL;
```

DELETE DATA

```
-- Delete specific record
DELETE FROM employees
WHERE name = 'John Smith';

-- Delete with conditions
DELETE FROM products
WHERE price < 100;

-- Delete multiple conditions
DELETE FROM employees
WHERE department = 'Sales' AND salary < 50000;

-- Delete all records (be very careful!)
DELETE FROM employees;  -- This removes all data!
```

ALTER TABLE

```
-- Add new column
ALTER TABLE employees
ADD COLUMN phone VARCHAR(20);

-- Modify column
ALTER TABLE employees
MODIFY COLUMN salary DECIMAL(12,2);

-- Drop column
ALTER TABLE employees
DROP COLUMN phone;

-- Add constraint
ALTER TABLE employees
ADD CONSTRAINT unique_email UNIQUE (email);
```

Common Data Types

```
-- Numeric types
INTEGER, INT           -- Whole numbers
DECIMAL(p,s)          -- Fixed-point numbers
FLOAT, DOUBLE         -- Floating-point numbers
```

```
-- String types
VARCHAR(n)             -- Variable-length string (max n chars)
CHAR(n)               -- Fixed-length string (exactly n chars)
TEXT                  -- Large text data

-- Date/Time types
DATE                  -- Date only (YYYY-MM-DD)
TIME                  -- Time only (HH:MM:SS)
DATETIME              -- Date and time
TIMESTAMP             -- Date and time with timezone

-- Other types
BOOLEAN               -- True/false values
BLOB                  -- Binary data
```

Constraints

```
CREATE TABLE orders (
    id INTEGER PRIMARY KEY,
    customer_id INTEGER NOT NULL,
    order_date DATE DEFAULT CURRENT_DATE,
    total DECIMAL(10,2) CHECK (total > 0),
    status VARCHAR(20) DEFAULT 'pending',

    FOREIGN KEY (customer_id) REFERENCES customers(id)
);

-- Common constraints:
-- PRIMARY KEY    - Unique identifier
-- NOT NULL       - Cannot be empty
-- UNIQUE         - Must be unique
-- DEFAULT        - Default value
-- CHECK          - Custom validation
-- FOREIGN KEY    - Reference to another table
```

Practical Examples

Example 1: Employee Management

```
-- Create employee table
CREATE TABLE employees (
    id INTEGER PRIMARY KEY AUTO_INCREMENT,
    name VARCHAR(100) NOT NULL,
    email VARCHAR(150) UNIQUE,
    department VARCHAR(50),
    salary DECIMAL(10,2),
    hire_date DATE DEFAULT CURRENT_DATE
);

-- Add employees
INSERT INTO employees (name, email, department, salary)
VALUES
    ('John Doe', 'john@company.com', 'Sales', 50000),
    ('Jane Smith', 'jane@company.com', 'IT', 65000);

-- Give raise to IT department
UPDATE employees
SET salary = salary * 1.1
WHERE department = 'IT';

-- Remove employees with low salary
DELETE FROM employees
WHERE salary < 30000;
```

Example 2: Product Inventory

```
-- Create products table
CREATE TABLE products (
    id INTEGER PRIMARY KEY AUTO_INCREMENT,
    name VARCHAR(100) NOT NULL,
    price DECIMAL(8,2) NOT NULL,
    stock INTEGER DEFAULT 0,
    category VARCHAR(50)
);

-- Add products
INSERT INTO products (name, price, stock, category)
VALUES
    ('Laptop', 999.99, 50, 'Electronics'),
    ('Mouse', 29.99, 100, 'Electronics'),
    ('Desk', 299.99, 20, 'Furniture');

-- Update stock after sale
UPDATE products
SET stock = stock - 1
WHERE name = 'Laptop';

-- Discount electronics
UPDATE products
SET price = price * 0.9
WHERE category = 'Electronics';
```

Best Practices

1. Always use WHERE with UPDATE/DELETE

```
-- GOOD: Specific update
UPDATE employees SET salary = 55000 WHERE id = 1;

-- BAD: Updates all records
UPDATE employees SET salary = 55000;  -- Dangerous!
```

2. Backup before major operations

```
-- Create backup table
CREATE TABLE employees_backup AS
```

```
SELECT * FROM employees;

-- Then perform operations
DELETE FROM employees WHERE department = 'Sales';
```

3. Use transactions for multiple operations

```
BEGIN TRANSACTION;
UPDATE accounts SET balance = balance - 100 WHERE id = 1;
UPDATE accounts SET balance = balance + 100 WHERE id = 2;
COMMIT;  -- or ROLLBACK if something goes wrong
```

4. Use meaningful column names

```
-- GOOD: Clear names
CREATE TABLE customers (
    customer_id INTEGER,
    first_name VARCHAR(50),
    last_name VARCHAR(50),
    email_address VARCHAR(100)
);

-- AVOID: Unclear names
CREATE TABLE customers (
    c_id INTEGER,
    fn VARCHAR(50),
    ln VARCHAR(50),
    ea VARCHAR(100)
);
```