SQL Basics: Essential Commands and Their Python Equivalents

1. SELECT Command

The SELECT command is used to retrieve data from a database table.

SQL Syntax:

```sql
-- Basic SELECT
SELECT column1, column2 FROM table_name;

-- Select all columns
SELECT * FROM table_name;

-- Select with column alias
SELECT first_name AS name,
       salary AS annual_income
FROM employees;

-- Select with calculations
SELECT product_name,
       unit_price * quantity AS total_cost
FROM orders;
```

Python Equivalent (using pandas):

```python
import pandas as pd

# Basic SELECT
df[['column1', 'column2']]

# Select all columns
df

# Select with column alias
df.rename(columns={'first_name': 'name',
                   'salary': 'annual_income'})

# Select with calculations
df['total_cost'] = df['unit_price'] * df['quantity']
```

2. FROM Command

FROM specifies the table(s) from which to retrieve data.

SQL Syntax:

```sql
-- Single table
SELECT * FROM employees;

-- Multiple tables (JOIN)
SELECT employees.name, departments.dept_name
FROM employees
JOIN departments ON employees.dept_id = departments.id;

-- Table alias
SELECT e.name, d.dept_name
FROM employees AS e
JOIN departments AS d ON e.dept_id = d.id;
```

Python Equivalent:

```python
# Single table
employees_df

# Multiple tables (JOIN)
merged_df = pd.merge(
    employees_df,
    departments_df,
    left_on='dept_id',
    right_on='id'
)

# Working with merged data
merged_df[['name', 'dept_name']]
```

3. ORDER BY Command

ORDER BY sorts the result set based on specified columns.

SQL Syntax:

```sql
-- Simple ascending order
SELECT * FROM employees
ORDER BY salary;

-- Descending order
SELECT * FROM employees
ORDER BY salary DESC;

-- Multiple columns
SELECT * FROM employees
ORDER BY department ASC, salary DESC;

-- Order by column position
SELECT first_name, last_name, salary
FROM employees
ORDER BY 3 DESC;  -- Orders by salary
```

Python Equivalent:

```python
# Simple ascending order
df.sort_values('salary')

# Descending order
df.sort_values('salary', ascending=False)

# Multiple columns
df.sort_values(['department', 'salary'],
               ascending=[True, False])

# Reset index after sorting (optional)
df.sort_values('salary').reset_index(drop=True)
```

4. WHERE Command (Filtering)

WHERE filters rows based on specified conditions.

SQL Syntax:

```sql
-- Simple comparison
SELECT * FROM employees
WHERE salary > 50000;
```

```sql
-- Multiple conditions with AND
SELECT * FROM employees
WHERE salary > 50000
AND department = 'Sales';

-- Multiple conditions with OR
SELECT * FROM employees
WHERE department = 'Sales'
OR department = 'Marketing';

-- IN operator
SELECT * FROM employees
WHERE department IN ('Sales', 'Marketing', 'IT');

-- LIKE operator for pattern matching
SELECT * FROM employees
WHERE last_name LIKE 'S%';  -- Names starting with S

-- BETWEEN operator
SELECT * FROM employees
WHERE salary BETWEEN 40000 AND 60000;
```

Python Equivalent:

```python
# Simple comparison
df[df['salary'] > 50000]

# Multiple conditions with AND
df[(df['salary'] > 50000) &
   (df['department'] == 'Sales')]

# Multiple conditions with OR
df[df['department'].isin(['Sales', 'Marketing'])]

# IN operator
df[df['department'].isin(['Sales', 'Marketing', 'IT'])]

# LIKE operator (using str.startswith())
df[df['last_name'].str.startswith('S')]

# BETWEEN operator
df[(df['salary'] >= 40000) & (df['salary'] <= 60000)]
```

Practice Example:

Here's a complete example combining multiple concepts:

SQL Version:

```sql
SELECT
    e.first_name,
    e.last_name,
    d.department_name,
    e.salary
FROM
    employees e
JOIN
    departments d ON e.dept_id = d.id
WHERE
    e.salary > 50000
    AND d.department_name IN ('Sales', 'Marketing')
ORDER BY
    e.salary DESC,
    e.last_name ASC;
```

Python Version:

```python
# Merge datasets
result = pd.merge(
    employees_df,
    departments_df,
    left_on='dept_id',
    right_on='id'
)

# Apply filters and sorting
filtered_df = (
    result[
        (result['salary'] > 50000) &
        (result['department_name'].isin(['Sales', 'Marketing']))
    ]
    .sort_values(['salary', 'last_name'],
                 ascending=[False, True])
    [['first_name', 'last_name', 'department_name', 'salary']]
)
```

Remember:

1. SQL is not case-sensitive for keywords, but it's common practice to write SQL keywords in uppercase for readability
2. Always end SQL statements with a semicolon
3. In Python/pandas, remember to import pandas as pd at the start of your script
4. When using multiple conditions in pandas, wrap each condition in parentheses
5. Be careful with AND/OR operations in pandas: use & for AND, | for OR