SQL Language Cheat Sheet

BEEKEEPER STUDIO

SELECT Basics

Retrieve data from tables by specifying which columns to return. Use DISTINCT to remove duplicates and AS for aliases.

```sql
SELECT customer_id, name, email
FROM customers;

-- With alias and DISTINCT
SELECT DISTINCT country AS customer_country
FROM customers;
```

Key Points:

- Use specific columns instead of * for better performance
- DISTINCT removes duplicate rows
- AS keyword creates column aliases
- Column order in SELECT determines result order

WHERE Clause

Filter rows based on conditions using comparison operators. Combine multiple conditions with AND/OR.

```sql
SELECT name, price
FROM products
WHERE price > 100;

-- Multiple conditions
SELECT name, category
FROM products
WHERE category = 'Electronics'
  AND price <= 500;
```

Key Points:

- Filters rows before returning results
- Can combine multiple conditions with AND/OR
- String values require single quotes
- Operators: =, !=, <>, <, >, <=, >=

Logical Operators

Combine multiple conditions using AND (all must be true), OR (at least one true), and NOT (negates condition).

```sql
-- AND operator
SELECT * FROM orders
WHERE status = 'shipped' AND total > 100;

-- OR operator
SELECT * FROM customers
WHERE country = 'USA' OR country = 'Canada';
```

Key Points:

- AND has higher precedence than OR
- Use parentheses to control evaluation order
- NOT is equivalent to != or <>
- Combine operators for complex logic

ORDER BY

Sort query results in ascending (ASC) or descending (DESC) order. Sort by multiple columns for complex ordering.

```sql
-- Single column sort
SELECT name, price
FROM products
ORDER BY price DESC;

-- Multiple columns
SELECT name, category, price
FROM products
ORDER BY category ASC, price DESC;
```

Key Points:

- ASC is default if not specified
- Sort by multiple columns (left to right priority)
- Can sort by column not in SELECT list
- NULL values sorted first or last (DB-dependent)

LIMIT / OFFSET

Limit the number of rows returned and skip rows using OFFSET. Essential for pagination and working with large result sets.

```sql
-- First 10 rows
SELECT name, price
FROM products
ORDER BY price DESC
LIMIT 10;

-- Pagination (skip 20, get next 10)
SELECT name, price
FROM products
ORDER BY price DESC
LIMIT 10 OFFSET 20;
```

Key Points:

- Essential for pagination
- Combine with ORDER BY for consistent results
- OFFSET starts at 0 (first row)
- Note: SQL Server uses TOP instead of LIMIT

Pattern Matching (LIKE)

Match text patterns using wildcards: % matches any sequence of characters, _ matches a single character.

```sql
-- Starts with 'John'
SELECT name, email
FROM customers
WHERE name LIKE 'John%';

-- Contains 'smith'
SELECT name FROM customers
WHERE name LIKE '%smith%';

-- Exact pattern (5 chars, starts with 'A')
SELECT code FROM products
WHERE code LIKE 'A____';
```

Key Points:

- % matches zero or more characters
- _ matches exactly one character
- Case sensitivity depends on database collation
- Can use NOT LIKE for exclusion

Range & Set (BETWEEN, IN)

Filter by ranges using BETWEEN (inclusive) or match values in a list using IN. More efficient than multiple OR conditions.

```sql
-- BETWEEN (inclusive)
SELECT name, price
FROM products
WHERE price BETWEEN 50 AND 100;

-- IN operator
SELECT name, country
FROM customers
WHERE country IN ('USA', 'Canada', 'Mexico');
```

Key Points:

- BETWEEN includes both boundary values
- IN is cleaner than multiple OR conditions
- Can use IN with subqueries
- BETWEEN works with dates and strings

NULL Handling

Work with NULL values using IS NULL and IS NOT NULL. Use COALESCE to provide default values for NULL columns.

```sql
-- Find NULL values
SELECT name, phone
FROM customers
WHERE phone IS NULL;

-- Exclude NULL values
SELECT name, email
FROM customers
WHERE email IS NOT NULL;

-- Provide default value
SELECT name, COALESCE(phone, 'No phone') AS contact
FROM customers;
```

Key Points:

- Cannot use = or != with NULL
- Must use IS NULL or IS NOT NULL
- COALESCE returns first non-NULL argument
- NULL in math operations results in NULL

Aggregate Functions

Perform calculations on sets of rows: COUNT, SUM, AVG, MIN, MAX. Aggregates ignore NULL values (except COUNT(*)).

```
-- Basic aggregates
SELECT
  COUNT(*) AS total_orders,
  SUM(amount) AS total_revenue,
  AVG(amount) AS avg_order,
  MIN(amount) AS smallest,
  MAX(amount) AS largest
FROM orders;

-- COUNT variations
SELECT COUNT(*), COUNT(email),
       COUNT(DISTINCT country)
FROM customers;
```

Key Points:

- COUNT(*) includes all rows (even with NULL)
- COUNT(column) excludes NULL values
- COUNT(DISTINCT column) counts unique values
- SUM/AVG only work with numeric types

GROUP BY / HAVING

Group rows by column values and apply aggregate functions. Use HAVING to filter groups (like WHERE for groups).

```
-- Group by single column
SELECT country, COUNT(*) AS customer_count
FROM customers
GROUP BY country
ORDER BY customer_count DESC;

-- Multiple columns with HAVING
SELECT category, status, AVG(price) AS avg_price
FROM products
GROUP BY category, status
HAVING AVG(price) > 50;
```

Key Points:

- All non-aggregate SELECT columns must be in GROUP BY
- WHERE filters rows before grouping
- HAVING filters groups after aggregation
- Can use aggregate functions in HAVING clause

Joins (Quick Reference) SEE DETAILED GUIDE

Combine data from multiple tables based on related columns. INNER JOIN returns matches only, LEFT/RIGHT preserve one side.

```
-- INNER JOIN (most common)
SELECT orders.id, customers.name, orders.amount
FROM orders
INNER JOIN customers
  ON orders.customer_id = customers.id;

-- LEFT JOIN (all orders + matching customers)
SELECT orders.id, customers.name
FROM orders
LEFT JOIN customers
  ON orders.customer_id = customers.id;
```

Key Points:

- INNER JOIN is most common (can write as just JOIN)
- LEFT/RIGHT JOINs preserve all rows from one table
- Use ON clause to specify join condition
- See beekeeperstudio.io/sql-join-cheat-sheet for more

Subqueries

Queries within queries for complex filtering and calculations. Use with IN, EXISTS, or as scalar values.

```
-- Subquery with IN
SELECT name, price
FROM products
WHERE category_id IN (
  SELECT id FROM categories
  WHERE name = 'Electronics'
);

-- Subquery with EXISTS
SELECT name FROM customers c
WHERE EXISTS (
  SELECT 1 FROM orders o
  WHERE o.customer_id = c.id
);
```

Key Points:

- IN subquery returns multiple values
- EXISTS checks for existence (efficient)
- Scalar subquery returns single value
- Correlated subquery references outer query

Useful Resources

SQL Reference Guides

- **SQL JOIN Cheat Sheet**
  beekeeperstudio.io/sql-join-cheat-sheet
  Visual guide to all JOIN types with diagrams
- **SQL Tutorials & Guides**
  beekeeperstudio.io/blog
  In-depth tutorials on SQL fundamentals

Free Developer Tools

- **SQL Formatter**
  beekeeperstudio.io/formatter
  Format and beautify your SQL queries
- **SQL Data Generator**
  beekeeperstudio.io/tools/sql-data-generator
  Generate realistic test data for tables
- **More Free Tools**
  beekeeperstudio.io/tools
  JSON to SQL, syntax checker, and more

Database-Specific Guides

- **PostgreSQL Documentation**
  beekeeperstudio.io/db/postgres
- **MySQL Documentation**
  beekeeperstudio.io/db/mysql
- **15+ Databases Supported**
  beekeeperstudio.io/databases

About Beekeeper Studio

Beekeeper Studio Query Editor

Beekeeper Studio is a modern, open-source SQL editor and database manager designed to make working with databases easier and more enjoyable.

✨ **Open Source & Free Community Edition**
Available on GitHub with a GPL license. Free for personal and commercial use.

Key Features:

- Autocomplete for tables, columns, and keywords
- Visual table editor with data editing
- Query history and saved queries
- Support for 15+ databases (PostgreSQL, MySQL, SQLite, SQL Server, Oracle, and more)
- Cross-platform: Linux, macOS, and Windows
- Import/export data in multiple formats

Download & Learn More:

beekeeperstudio.io

GitHub: github.com/beekeeper-studio/beekeeper-studio