DataLens Query Language Reference

Document ID: PRD-DL-010 **Last Updated:** 2024-02-15 **Owner:** DataLens Engineering **Classification:** Public

---

Overview

DataLens Query Language (DQL) is a SQL-like language for querying data from connected sources. This reference covers syntax, functions, and examples.

Basic Syntax

SELECT Statement

```
SELECT column1, column2, ...
FROM data_source
WHERE conditions
GROUP BY columns
HAVING aggregate_conditions
ORDER BY columns
LIMIT n
```

Simple Example

```
SELECT
    customer_name,
    SUM(order_total) as total_spent
FROM orders
WHERE order_date >= '2024-01-01'
GROUP BY customer_name
ORDER BY total_spent DESC
LIMIT 10
```

Data Types

Type	Description	Example
STRING	Text values	'Hello World'
NUMBER	Numeric values	42, 3.14
BOOLEAN	True/false	true, false
DATETIME	Date and time	'2024-01-15 10:30:00'
DATE	Date only	'2024-01-15'
TIME	Time only	'10:30:00'
NULL	Null value	null

Operators

Comparison Operators

Operator	Description	Example
=	Equal	`status = 'active'`
!= or <>	Not equal	`status != 'deleted'`
>	Greater than	`amount > 100`
<	Less than	`amount < 100`
>=	Greater or equal	`age >= 18`
<=	Less or equal	`age <= 65`
BETWEEN	Range	`age BETWEEN 18 AND 65`
IN	List match	`status IN ('active', 'pending')`
LIKE	Pattern match	`name LIKE '%Smith%'`
IS NULL	Null check	`email IS NULL`
IS NOT NULL	Not null check	`email IS NOT NULL`

Logical Operators

Operator	Description	Example
AND	Both conditions	`a > 1 AND b < 10`
OR	Either condition	`a > 1 OR b < 10`
NOT	Negation	`NOT status = 'deleted'`

Arithmetic Operators

Operator	Description	Example
+	Addition	`price + tax`
-	Subtraction	`total - discount`
*	Multiplication	`quantity * price`
/	Division	`total / count`
%	Modulo	`id % 10`

Aggregate Functions

Basic Aggregates

Function	Description	Example
COUNT(*)	Count rows	`COUNT(*)`
COUNT(col)	Count non-null	`COUNT(email)`
COUNT(DISTINCT col)	Count unique	`COUNT(DISTINCT customer_id)`
SUM(col)	Sum values	`SUM(amount)`
AVG(col)	Average	`AVG(price)`
MIN(col)	Minimum	`MIN(created_at)`
MAX(col)	Maximum	`MAX(order_total)`

Statistical Functions

Function	Description	Example
STDDEV(col)	Standard deviation	`STDDEV(values)`
VARIANCE(col)	Variance	`VARIANCE(values)`
PERCENTILE(col, p)	Percentile	`PERCENTILE(response_time, 0.95)`
MEDIAN(col)	Median value	`MEDIAN(salary)`

Date/Time Functions

Current Date/Time

Function	Description	Example
NOW()	Current timestamp	`NOW()`
CURRENT_DATE()	Current date	`CURRENT_DATE()`
CURRENT_TIME()	Current time	`CURRENT_TIME()`

Date Extraction

Function	Description	Example
YEAR(dt)	Extract year	`YEAR(created_at)`
MONTH(dt)	Extract month	`MONTH(created_at)`
DAY(dt)	Extract day	`DAY(created_at)`
HOUR(dt)	Extract hour	`HOUR(created_at)`
MINUTE(dt)	Extract minute	`MINUTE(created_at)`
DAYOFWEEK(dt)	Day of week (1-7)	`DAYOFWEEK(created_at)`
DAYOFYEAR(dt)	Day of year	`DAYOFYEAR(created_at)`
WEEK(dt)	Week number	`WEEK(created_at)`
QUARTER(dt)	Quarter (1-4)	`QUARTER(created_at)`

Date Arithmetic

Function	Description	Example
DATE_ADD(dt, interval)	Add to date	`DATE_ADD(NOW(), INTERVAL 7 DAY)`
DATE_SUB(dt, interval)	Subtract from date	`DATE_SUB(NOW(), INTERVAL 1 MONTH)`
DATEDIFF(dt1, dt2)	Days between	`DATEDIFF(end_date, start_date)`
TIMESTAMPDIFF(unit, dt1, dt2)	Difference in unit	`TIMESTAMPDIFF(HOUR, start, end)`

Date Formatting

```
DATE_FORMAT(created_at, '%Y-%m-%d')  -- 2024-01-15
DATE_FORMAT(created_at, '%b %d, %Y')  -- Jan 15, 2024
DATE_FORMAT(created_at, '%H:%i')      -- 14:30
```

Time Buckets

```
-- Group by hour
TIME_BUCKET('1 hour', timestamp) as hour
```

```
-- Group by day
TIME_BUCKET('1 day', timestamp) as day

-- Group by week
TIME_BUCKET('1 week', timestamp) as week
```

String Functions

Function	Description	Example
CONCAT(s1, s2, ...)	Concatenate	`CONCAT(first, ' ', last)`
UPPER(s)	Uppercase	`UPPER(name)`
LOWER(s)	Lowercase	`LOWER(email)`
TRIM(s)	Remove whitespace	`TRIM(input)`
LENGTH(s)	String length	`LENGTH(description)`
SUBSTRING(s, start, len)	Extract substring	`SUBSTRING(phone, 1, 3)`
REPLACE(s, from, to)	Replace text	`REPLACE(url, 'http', 'https')`
SPLIT(s, delimiter)	Split to array	`SPLIT(tags, ',')`
REGEXP_MATCH(s, pattern)	Regex match	`REGEXP_MATCH(email, '@.*')`

Conditional Functions

CASE Expression

```
SELECT
    order_id,
    CASE
        WHEN total >= 1000 THEN 'Large'
        WHEN total >= 100 THEN 'Medium'
        ELSE 'Small'
    END as order_size
FROM orders
```

COALESCE

```sql
-- Return first non-null value
SELECT COALESCE(preferred_name, first_name, 'Unknown') as display_name
FROM users
```

NULLIF

```sql
-- Return NULL if values are equal
SELECT total / NULLIF(count, 0) as average  -- Avoids division by zero
FROM stats
```

IF

```sql
SELECT IF(status = 'active', 'Yes', 'No') as is_active
FROM users
```

Window Functions

Syntax

```sql
function() OVER (
    PARTITION BY column
    ORDER BY column
    ROWS/RANGE frame_specification
)
```

Common Window Functions

Function	Description
ROW_NUMBER()	Sequential row number
RANK()	Rank with gaps
DENSE_RANK()	Rank without gaps
LAG(col, n)	Previous row value
LEAD(col, n)	Next row value
FIRST_VALUE(col)	First value in window
LAST_VALUE(col)	Last value in window
SUM() OVER ()	Running sum
AVG() OVER ()	Running average

Examples

```sql
-- Row numbers per customer
SELECT
    customer_id,
    order_date,
    ROW_NUMBER() OVER (PARTITION BY customer_id ORDER BY order_date) as order_num
FROM orders

-- Running total
SELECT
    date,
    revenue,
    SUM(revenue) OVER (ORDER BY date) as cumulative_revenue
FROM daily_sales

-- Compare to previous period
SELECT
    date,
    revenue,
    LAG(revenue, 7) OVER (ORDER BY date) as revenue_7_days_ago,
    revenue - LAG(revenue, 7) OVER (ORDER BY date) as change
FROM daily_sales
```

Joins

Inner Join

```sql
SELECT o.*, c.name
FROM orders o
INNER JOIN customers c ON o.customer_id = c.id
```

Left Join

```sql
SELECT c.*, COUNT(o.id) as order_count
FROM customers c
LEFT JOIN orders o ON c.id = o.customer_id
GROUP BY c.id
```

Multiple Joins

```sql
SELECT
    o.id,
    c.name as customer,
    p.name as product
FROM orders o
JOIN customers c ON o.customer_id = c.id
JOIN products p ON o.product_id = p.id
```

Subqueries

In WHERE Clause

```sql
SELECT * FROM customers
WHERE id IN (
    SELECT customer_id
    FROM orders
    WHERE total > 1000
)
```

In FROM Clause

```sql
SELECT segment, AVG(total_spent)
FROM (
    SELECT
        customer_id,
        SUM(total) as total_spent,
        CASE WHEN SUM(total) > 1000 THEN 'High' ELSE 'Low' END as segment
    FROM orders
    GROUP BY customer_id
) subquery
GROUP BY segment
```

Variables

Dashboard Variables

Reference dashboard variables with `${variable}`:

```
SELECT * FROM orders
WHERE region = '${region}'
AND status IN (${status})
AND created_at >= '${__from}'
AND created_at <= '${__to}'
```

Built-in Variables

Variable	Description
${__from}	Dashboard start time
${__to}	Dashboard end time
${__interval}	Auto-calculated interval
${__user}	Current user

Time Filters

Time Range Macros

```
-- Automatic time filter
SELECT * FROM events
WHERE $__timeFilter(timestamp)

-- Equivalent to:
-- WHERE timestamp >= '2024-01-01' AND timestamp <= '2024-01-31'
```

Relative Time

```
-- Last 7 days
WHERE created_at >= NOW() - INTERVAL 7 DAY

-- This month
WHERE YEAR(created_at) = YEAR(NOW())
  AND MONTH(created_at) = MONTH(NOW())

-- Year to date
WHERE created_at >= DATE_TRUNC('year', NOW())
```

Performance Tips

1. **Filter early:** Apply WHERE clauses before JOINs
2. **Limit results:** Use LIMIT for exploratory queries
3. **Index columns:** Filter on indexed columns when possible
4. **Avoid SELECT *:** Select only needed columns
5. **Use aggregates wisely:** Pre-aggregate when possible

Related Documents: Getting Started (PRD-DL-001), Dashboard Creation Guide (PRD-DL-005), Alerting Guide (PRD-DL-015)