SQL Syntax.
DML

Overview

- Lexical Structure
- Identifiers and Key Words
- Operators
- DML
- Inserting Data
- Updating Data
- Deleting Data

Lexical Structure

- SQL input consists of a sequence of *commands*.

Lexical Structure

- SQL input consists of a sequence of *commands*.
- A command is composed of a sequence of *tokens*, terminated by a semicolon (“;”)

Lexical Structure

- SQL input consists of a sequence of *commands*.
- A command is composed of a sequence of *tokens*, terminated by a semicolon (“;”)
- The end of the input stream also terminates a command

Lexical Structure

- A token can be a *key word*, an *identifier*, a *quoted identifier*, a *literal* (or constant), or a special character symbol

Lexical Structure

- A token can be a *key word*, an *identifier*, a *quoted identifier*, a *literal* (or constant), or a special character symbol

- Which tokens are valid depends on the syntax of the particular command

Lexical Structure

```
SELECT * FROM MY_TABLE;
UPDATE MY_TABLE SET A = 5;
INSERT INTO MY_TABLE VALUES (3, 'hi there');
```

Lexical Structure

- more than one command can be on a line

Lexical Structure

- more than one command can be on a line
- commands can usefully be split across lines

Lexical Structure

- more than one command can be on a line
- commands can usefully be split across lines
- comments can occur in SQL input. They are not tokens, they are effectively equivalent to whitespace

Identifiers and Key Words

- SQL identifiers and key words must begin with a letter or an underscore

Identifiers and Key Words

- SQL identifiers and key words must begin with a letter or an underscore
- SQL standard will not define a key word that contains digits or starts or ends with an underscore

Identifiers and Key Words

- SQL identifiers and key words must begin with a letter or an underscore
- SQL standard will not define a key word that contains digits or starts or ends with an underscore
- maximum identifier length is 63 bytes

Identifiers and Key Words

- Key words and unquoted identifiers are case insensitive

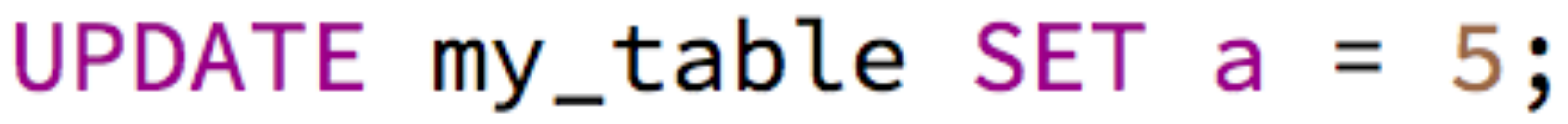
```
UPDATE MY_TABLE SET A = 5;

uPDaTE my_TabLE SeT a = 5;

UPDATE my_table SET a = 5;
```

Identifiers and Key Words

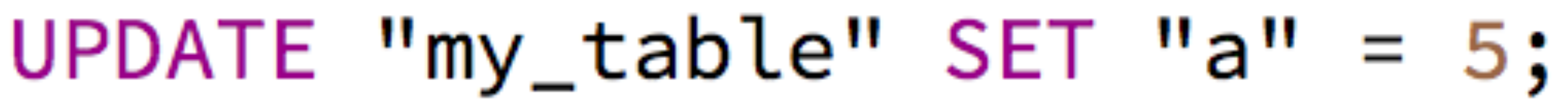
- It is better to write key words in upper and identifiers in lower case



```sql
UPDATE my_table SET a = 5;
```

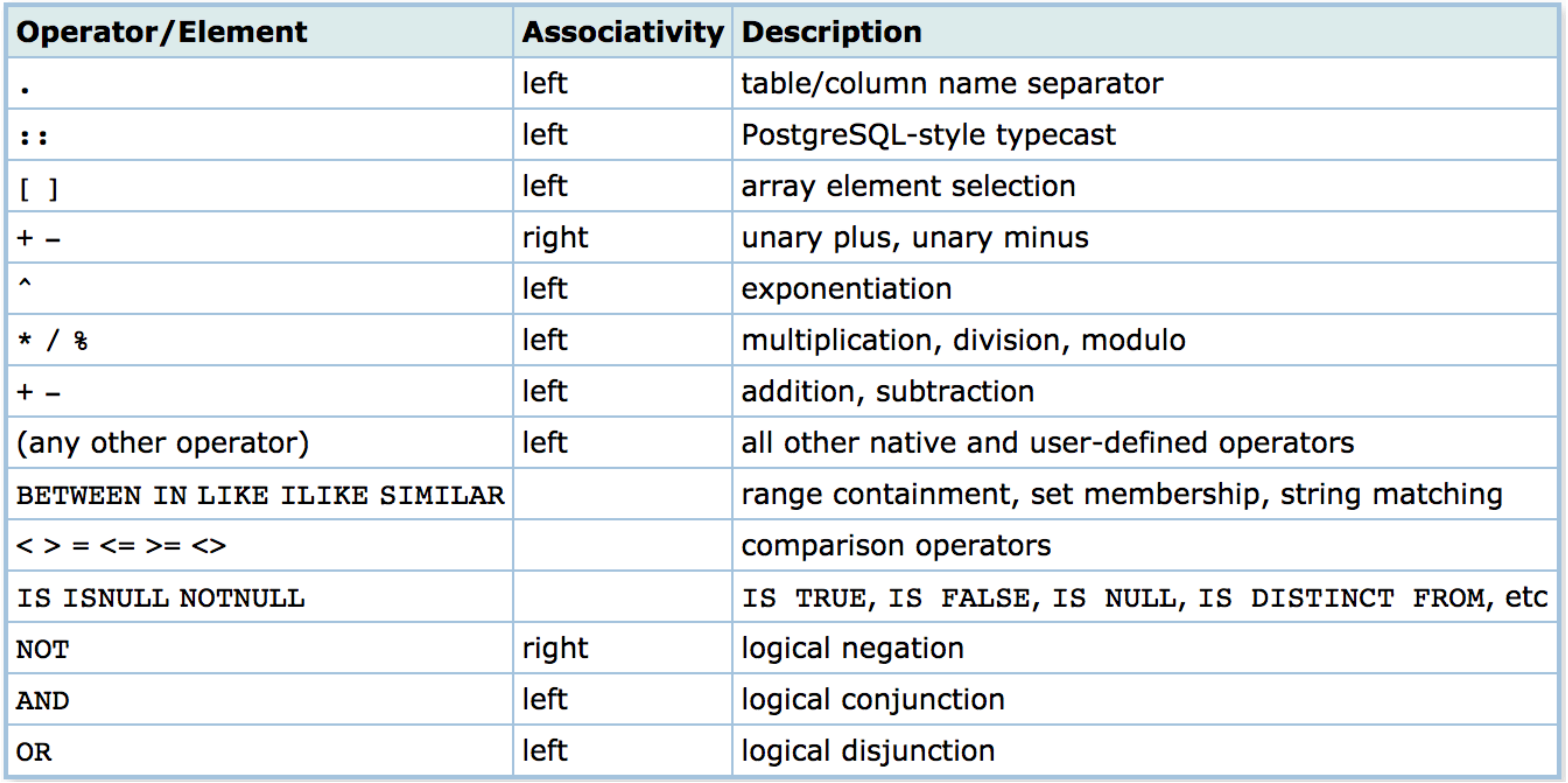
Identifiers and Key Words

- There is a second kind of identifier: the *delimited identifier* or *quoted identifier*



```
UPDATE "my_table" SET "a" = 5;
```

Operator Precedence (highest to lowest)



Operator/Element	**Associativity**	**Description**
`.`	left	table/column name separator
`::`	left	PostgreSQL-style typecast
`[ ]`	left	array element selection
`+ -`	right	unary plus, unary minus
`^`	left	exponentiation
`* / %`	left	multiplication, division, modulo
`+ -`	left	addition, subtraction
(any other operator)	left	all other native and user-defined operators
`BETWEEN IN LIKE ILIKE SIMILAR`		range containment, set membership, string matching
`< > = <= >= <>`		comparison operators
`IS ISNULL NOTNULL`		`IS TRUE`, `IS FALSE`, `IS NULL`, `IS DISTINCT FROM`, etc
`NOT`	right	logical negation
`AND`	left	logical conjunction
`OR`	left	logical disjunction

Comments

```
-- This is a standard SQL comment
```

```
/* multiline comment
 * with nesting: /* nested block comment */
 */
```

DML

- DML statements are SQL statements that manipulate data.

DML

- DML statements are SQL statements that manipulate data.
- DML stands for Data Manipulation Language.

DML

- DML statements are SQL statements that manipulate data.
- DML stands for Data Manipulation Language.
- The SQL statements that are in the DML class are INSERT, UPDATE and DELETE.

DML

- DML statements are SQL statements that manipulate data.
- DML stands for Data Manipulation Language.
- The SQL statements that are in the DML class are INSERT, UPDATE and DELETE.
- Some people also lump the SELECT statement in the DML classification.

Inserting Data

- When a table is created, it contains no data

Inserting Data

- When a table is created, it contains no data
- The first thing to do before a database can be of much use is to insert data

Inserting Data

- When a table is created, it contains no data
- The first thing to do before a database can be of much use is to insert data
- Even if you know only some column values, a complete row must be created

Inserting Data

- INSERT — create new rows in a table

```
INSERT INTO table_name [ AS alias ] [ ( column_name [, ...] ) ]
    { DEFAULT VALUES | VALUES ( { expression | DEFAULT } [, ...] ) [, ...] | query }
```

Parameters

- *table_name* - The name (optionally schema-qualified) of an existing table

Parameters

- *table_name* - The name (optionally schema-qualified) of an existing table
- *alias* - A substitute name for *table_name*. When an alias is provided, it completely hides the actual name of the table.

Parameters

- *table_name* - The name (optionally schema-qualified) of an existing table
- *alias* - A substitute name for *table_name*. When an alias is provided, it completely hides the actual name of the table.
- *column_name* - The name of a column in the table named by *table_name*.

Parameters

- *DEFAULT VALUES* - All columns will be filled with their default values

Parameters

- *DEFAULT VALUES* - All columns will be filled with their default values
- *expression* - An expression or value to assign to the corresponding column

Parameters

- *DEFAULT* - The corresponding column will be filled with its default value

Parameters

- *DEFAULT* - The corresponding column will be filled with its default value
- *query* - A query (SELECT statement) that supplies the rows to be inserted

Parameters

- The target column names can be listed in any order

Parameters

- The target column names can be listed in any order
- If no list of column names is given at all, the default is all the columns of the table in their declared order

Parameters

- The target column names can be listed in any order
- If no list of column names is given at all, the default is all the columns of the table in their declared order
- or the first N column names, if there are only N columns supplied by the VALUES clause or query

Parameters

- Each column not present in the column list will be filled with a default value, either its declared default value or null if there is none.

Parameters

- Each column not present in the column list will be filled with a default value, either its declared default value or null if there is none.
- If the expression for any column is not of the correct data type, automatic type conversion will be attempted.

Parameters

- Each column not present in the column list will be filled with a default value, either its declared default value or null if there is none.
- If the expression for any column is not of the correct data type, automatic type conversion will be attempted.
- You must have INSERT privilege on a table in order to insert into it.

Examples

```
INSERT INTO films VALUES
            ('UA502', 'Bananas', 105, '1971-07-13', 'Comedy', '82 minutes');
```

```
INSERT INTO films (code, title, did, date_prod, kind)
    VALUES ('T_601', 'Yojimbo', 106, '1961-06-16', 'Drama');
```

Examples

```
INSERT INTO films (code, title, did, date_prod, kind)
    VALUES ('T_601', 'Yojimbo', 106, DEFAULT, 'Drama');
```

```
INSERT INTO films DEFAULT VALUES;
```

Examples

```
INSERT INTO films (code, title, did, date_prod, kind) VALUES
    ('B6717', 'Tampopo', 110, '1985-02-10', 'Comedy'),
    ('HG120', 'The Dinner Game', 140, DEFAULT, 'Comedy');
```

Examples

```
INSERT INTO films
    SELECT * FROM tmp_films
        WHERE date_prod < '2004-05-07';
```

Updating Data

- You can update individual rows, all the rows in a table, or a subset of all rows

Updating Data

- You can update individual rows, all the rows in a table, or a subset of all rows
- Each column can be updated separately. The other columns are not affected

Updating Data

- You can update individual rows, all the rows in a table, or a subset of all rows
- Each column can be updated separately. The other columns are not affected
- To update existing rows, use the UPDATE command

Updating Data

- UPDATE — update rows of a table

Updating Data

- UPDATE — update rows of a table
- UPDATE changes the values of the specified columns in all rows that satisfy the condition

Updating Data

- UPDATE — update rows of a table
- UPDATE changes the values of the specified columns in all rows that satisfy the condition
- Only the columns to be modified need be mentioned in the SET clause

Updating Data

- UPDATE — update rows of a table
- UPDATE changes the values of the specified columns in all rows that satisfy the condition
- Only the columns to be modified need be mentioned in the SET clause
- Columns not explicitly modified retain their previous values.

Update syntax

```
UPDATE [ ONLY ] table_name [ * ] [ [ AS ] alias ]
    SET { column_name = { expression | DEFAULT } |
          ( column_name [, ...] ) = [ ROW ] ( { expression | DEFAULT } [, ...] ) |
          ( column_name [, ...] ) = ( sub-SELECT )
        } [, ...]
    [ FROM from_list ]
    [ WHERE condition ]
```

Parameters

- *table_name* - The name (optionally schema-qualified) of the table to update

Parameters

- *table_name* - The name (optionally schema-qualified) of the table to update
- If `ONLY` is specified before the table name, matching rows are updated in the named table only

Parameters

- *table_name* - The name (optionally schema-qualified) of the table to update
- If `ONLY` is specified before the table name, matching rows are updated in the named table only
- If `ONLY` is not specified, matching rows are also updated in any tables inheriting from the named table

Parameters

- *table_name* - The name (optionally schema-qualified) of the table to update
- If `ONLY` is specified before the table name, matching rows are updated in the named table only
- `ONLY` is not specified, matching rows are also updated in any tables inheriting from the named table
- Optionally, * can be specified after the table name to explicitly indicate that descendant tables are included

Parameters

- *alias* - A substitute name for the target table. When an alias is provided, it completely hides the actual name of the table.

Parameters

- *alias* - A substitute name for the target table. When an alias is provided, it completely hides the actual name of the table.

- *column_name* - The name of a column in the table named by `table_name`

Parameters

- *alias* - A substitute name for the target table. When an alias is provided, it completely hides the actual name of the table.
- *column_name* - The name of a column in the table named by `table_name`
- *expression* - An expression to assign to the column. The expression can use the old values of this and other columns in the table

Parameters

- *DEFAULT* - Set the column to its default value (which will be NULL if no specific default expression has been assigned to it)

Parameters

- *DEFAULT* - Set the column to its default value (which will be NULL if no specific default expression has been assigned to it)

- *sub-SELECT* - A `SELECT` sub-query that produces as many output columns as are listed in the parenthesized column list preceding it. The sub-query must yield no more than one row when executed.

Parameters

- *from_list* - A list of table expressions, allowing columns from other tables to appear in the `WHERE` condition and the update expressions.

Parameters

- *from_list* - A list of table expressions, allowing columns from other tables to appear in the `WHERE` condition and the update expressions.

- *condition* - An expression that returns a value of type `boolean`. Only rows for which this expression returns `true` will be updated.

Examples

```sql
UPDATE films SET kind = 'Dramatic';
```

```sql
UPDATE films SET kind = 'Dramatic' WHERE kind = 'Drama';
```

Examples

```
UPDATE weather SET temp_lo = temp_lo+1, prcp = DEFAULT
  WHERE city = 'San Francisco' AND date = '2003-07-03';
```

```
UPDATE weather SET (temp_lo, prcp) = (temp_lo+1, DEFAULT)
  WHERE city = 'San Francisco' AND date = '2003-07-03';
```

Examples

```
UPDATE employees SET sales_count = sales_count + 1 FROM accounts
  WHERE accounts.name = 'Acme Corporation'
  AND employees.id = accounts.sales_person;
```

Examples

```
UPDATE employees SET sales_count = sales_count + 1 WHERE id =
  (SELECT sales_person FROM accounts WHERE name = 'Acme Corporation');
```

Examples

```
UPDATE accounts SET (contact_first_name, contact_last_name) =
    (SELECT first_name, last_name FROM salesmen
     WHERE salesmen.id = accounts.sales_id);
```

Deleting Data

- You can only remove entire rows from a table

Deleting Data

- You can only remove entire rows from a table
- Removing rows can only be done by specifying conditions that the rows to be removed have to match

Deleting Data

- You can only remove entire rows from a table
- Removing rows can only be done by specifying conditions that the rows to be removed have to match
- You can remove all rows in the table at once

Deleting Data

- DELETE — delete rows of a table

Deleting Data

- DELETE — delete rows of a table
- `DELETE` deletes rows that satisfy the `WHERE` clause from the specified table.

Deleting Data

- DELETE — delete rows of a table
- `DELETE` deletes rows that satisfy the `WHERE` clause from the specified table.
- If the `WHERE` clause is absent, the effect is to delete all rows in the table.

Delete syntax

```
DELETE FROM [ ONLY ] table_name [ * ] [ [ AS ] alias ]
    [ USING using_list ]
    [ WHERE condition ]
    [ RETURNING * | output_expression [ [ AS ] output_name ] [, ...] ]
```

Parameters

- *using_list* - A list of table expressions, allowing columns from other tables to appear in the `WHERE` condition.

Parameters

- *using_list* - A list of table expressions, allowing columns from other tables to appear in the `WHERE` condition.
- *condition* - An expression that returns a value of type `boolean`. Only rows for which this expression returns `true` will be deleted.

Examples

```sql
DELETE FROM films;

DELETE FROM films WHERE kind <> 'Musical';
```

Examples

```sql
DELETE FROM films USING producers
  WHERE producer_id = producers.id AND producers.name = 'foo';
```

Returning Data From Modified Rows

- Sometimes it is useful to obtain data from modified rows while they are being manipulated.

Returning Data From Modified Rows

- Sometimes it is useful to obtain data from modified rows while they are being manipulated.

- The `INSERT`, `UPDATE`, and `DELETE` commands all have an optional `RETURNING` clause that supports this.

Returning Data From Modified Rows

- Sometimes it is useful to obtain data from modified rows while they are being manipulated.
- The `INSERT`, `UPDATE`, and `DELETE` commands all have an optional `RETURNING` clause that supports this.
- Use of `RETURNING` avoids performing an extra database query to collect the data

Returning Data From Modified Rows

- The allowed contents of a `RETURNING` clause are the same as a `SELECT` command's output list

Returning Data From Modified Rows

- The allowed contents of a `RETURNING` clause are the same as a `SELECT` command's output list
- A common shorthand is `RETURNING *`, which selects all columns of the target table in order.

Examples

```
INSERT INTO users (firstname, lastname)
    VALUES ('Joe', 'Cool')
    RETURNING id;
```

Examples

```
UPDATE products SET price = price * 1.10
  WHERE price <= 99.99
  RETURNING name, price AS new_price;
```

Examples

```
DELETE FROM products
  WHERE obsoletion_date = 'today'
  RETURNING *;
```

Questions?