

Introduction to SQL

Author: Joshua Thompson

Created: October 29, 2008

The LINGUIST List

Background

- SQL is a vendor-neutral language for working with data stored in a database
- It is a simple yet powerful language that uses a very English-like syntax
- All major DBMS products support some form of SQL, though they vary in how fully they support the standard
- Most DBMS products also add their own vendor-specific extensions

Databases

- A **database** is an organized collection of data. Think of it as a well-labeled filing cabinet
- A database consists of one or more **tables**..

Tables

- A table holds data on one specific type of data, for example customers or invoices.
- Think of the table as one drawer in the filing cabinet

Rows

- A table consists of zero or more **rows** (also called **records**)
- A row represents one specific data item in the collection, such as a single customer or a single invoice.
- Think of a row as one folder in the drawer.

Columns

- A table row consists of one or more **columns**.
- A column represents a specific piece of information about the data, such as the customer's name or the due date of an invoice
- Every column has a well-defined **data type** that determines what kind of data is stored in that column. For example a column may be defined to hold only dates, or only integers.

Columns

- Within a single table all rows have the same type and number of columns.
- You can also imagine the rows & columns of a table as being like a spreadsheet, but with more stringent restrictions on what kinds of data can go into the cells.

Date Types

- **char/varchar/nchar/nvarchar** – stores text strings. The ones beginning with 'n' are Oracle-specific types that store Unicode text.
- **int** – stores integers. Called **number** in Oracle
- **float** - stores decimal numbers. Also called **number** in Oracle

Data Types

- **Date** – Stores datetime values. By default only shows date as “DD-MON-YY”.
- When giving values in a SQL statement (such as in INSERT or UPDATE) you must put single quotes (') around all **char** and **date** values. Numeric values should not be quoted.
- Unicode strings are quoted with n"
- NULL is always NULL.

Example Table

Name	Null?	Type
CUSTOMER_ID	NOT NULL	NUMBER (38) ↑
FIRST_NAME	NOT NULL	NVARCHAR (50) ↑
LAST_NAME	NOT NULL	NVARCHAR (50) ↑
CITY	NOT NULL	NVARCHAR (50) ↑
STATE	NOT NULL	CHAR (2) ↑
ZIP_CODE	NOT NULL	CHAR (9) ↑
AGE	NOT NULL	NUMBER (38) ↑

Retrieving data: SELECT

- The most commonly used SQL statement is the **SELECT** statement, which is used to retrieve data from one or more tables in the database.

```
SELECT <column list>
FROM   <table name>
[ WHERE <conditions> ]
[ GROUP BY <column list> ]
[ ORDER BY <column list> ]
```

- The portions inside square brackets are optional.

Retrieving data: SELECT

- The <column list> is a comma-separated list of column names you want to retrieve, or '*' to retrieve all columns
- The **WHERE** clause can be used to restrict the operation to only rows that match certain conditions. If you don't specify a **WHERE** clause then all rows in the table are returned.
- The following example retrieves the first_name and last_name columns of all customers in Michigan:

```
SELECT first_name, last_name
FROM customer
WHERE state = 'MI'
```

Ordering Results: ORDER BY

- The **ORDER BY** clause can be used to alter the order of the rows returned, instead of the more or less random order that is the default.
- You can order the result by one or more columns. If you specify multiple columns, the ordering is done by the first column, then the second, etc.
- For example, to order our customer list by name:

```
SELECT    first_name, last_name
FROM      customer
WHERE     state = 'MI'
ORDER BY  last_name, first_name
```

Ordering Results: ORDER BY

- By default ordering is done in ascending order, but you can reverse this by specifying **DESC** (for “descending”) after the column name.
- For example, to retrieve a list of all customers sorted by age, with oldest customers first:

```
SELECT    first_name, last_name
FROM      customer
WHERE     state = 'MI'
ORDER BY  age DESC
```

Adding rows: INSERT

- Inserting new rows into a table is accomplished with the INSERT statement:

```
INSERT INTO <table_name>  
(<column_list>) ↑  
VALUES  
(<values>) ↑
```

- This creates a new row, filling in the columns in <column_list> with the values in <value_list>. There is a 1:1 mapping between the column and value lists.

Adding rows: INSERT

- Unless you know what you are doing you should always fill in all columns
- Example:

```
INSERT INTO customer
(customer_id, first_name,
last_name, city, state,
zip_code, age)
VALUES
(1, n'John', n'Doe', n'Somewhere',
'MI', '48123', 42)
```

Modifying rows: UPDATE

- Modifying existing rows in a table is accomplished with the UPDATE statement:

```
UPDATE <table_name>  
SET    <column_updates>  
[ WHERE <condition> ]
```

- The <column_updates> is a comma-separated list of clauses in the form “<column> = <new value>”.

Modifying rows: UPDATE

- The WHERE clause is the same format as for **SELECT** and determines what row or rows get updated.
- Although the WHERE clause is optional it should **ALWAYS** be included. If you do not include one, you will modify ***every row in the table!***
- Example:

```
UPDATE orders
SET      status = 'PAID'
WHERE   order_number = '1001-402'
```