

DBS211 Introduction to Structured Query Language (SQL)

Dr. Rani Gnanaolivu

rani.gnanaolivu@senecacollege.ca

SENECA COLLEGE

Agenda

- ➔ • SQL and SQL Standardization
 - The basic commands and functions of SQL
 - How to use SQL to query a database to extract useful information (The SELECT statement)
- Sub-Languages of SQL
- The Basics of SELECT
- Aliases (Field and Table Aliases)
- INSERT - Inserting New Data Rows
- UPDATE - Updating Existing Data
- DELETE - Deleting Existing Data

Introduction to SQL

SQL: Structured Query Language

- Pronounced Sea - Quel

Universal Language used specifically for communicating with databases

SQL functions fit into **three broad categories:**

- DDL - Data definition language
- DML - Data manipulation language
- TCL - Transaction Control Language

SQL – 3 sub-categories

DDL - Data Definition Language

- Create database objects such as tables
- Commands to define access rights to those database objects
- Essentially working with the **STRUCTURE** of the database design

DML - Data Manipulation Language

- Commands to work with the **DATA** stored in the database
- Includes commands to insert, update, delete, and retrieve data within the database tables objects

TCL - Transaction Control Language

- Includes commands to ensure the integrity of the database
- Working with Multi-Step procedures to ensure everything that is supposed to happen, actually happens.

Introduction to SQL

5

SQL is relatively easy to learn

- But can get complicated quickly when attention to detail is applied

Basic command set has a vocabulary of less than 100 words

Sample vocabulary:

- | | |
|--------------------------------|-------------------------|
| – CREATE COLLECTION.... | DROP TABLE |
| – CREATE TABLE.... | DROP VIEW |
| – CREATE VIEW.... | ALTER TABLE |
| – SELECT * FROM | GRANT |
| – INSERT INTO | REVOKE |
| – UPDATE | |
| – DELETE | |

SQL Dialects

American National Standards Institute (ANSI)

- prescribes a standard SQL

Several SQL dialects exist

- DB2, Oracle, MySQL, Post-Gres SQL, MS Access, and MS-SQL etc
- Be careful when researching online to find solutions that work in the DBMS you are currently using

The SELECT command

7

Introducing the SELECT command general syntax

```
SELECT <comma separated field list *>  
      FROM <tablename(s) *>  
      WHERE <condition(s) using logical  
operators>  
      ORDER BY <comma separated field list> ;
```

* - means required part

Sample Table: PART

PART NUMBER	PART DESC	ON HAND	CLASS	WAREHOUSE	PRICE
AX12	Iron	104	HW	3	23.95
AZ52	Dartboard	20	SG	2	12.95
BA74	Basketball	40	SG	1	29.95
BH22	Cornpopper	95	HW	3	24.95
BT04	Gas Grill	11	AP	2	149.99
BZ66	Washer	52	AP	3	399.99
CA14	Griddle	78	HW	3	39.99
CB03	Bike	44	SG	1	299.99
CX11	Blender	112	HW	3	22.95
CZ81	Treadmill	68	SG	2	349.99

Listing Table Rows

At a minimum, must specify what you want to select and where you want to select it from

```
SELECT part_number  
FROM part;
```

RESULT

PART NUMBER
AX12
AZ52
BA74
BH22
BT04
BZ66
CA14
CB03
CX11
CZ81

Listing All Table Rows and Columns

Asterisk can be used as **wildcard** character to list all fields

```
SELECT *  
FROM part;
```

RESULT

Returns **all** rows and **all** fields - so basically the entire table

Selecting Rows with Comparison Operators

Select partial table contents by placing conditions on rows (records) to be included in output

- Add conditional restrictions to the SELECT statement, using **WHERE** clause

```
SELECT *  
      FROM part  
      WHERE on_hand >  
90;
```

RESULT - Returns **all** fields but only those records where on_hand > 90

PART NUMBER	PART DESC	ON HAND	CLASS	WAREHOUSE	PRICE
AX12	Iron	104	HW	3	23.95
BH22	Cornpopper	95	HW	3	24.95
CX11	Blender	112	HW	3	22.95

Comparison Operators

TABLE 6.7 COMPARISON OPERATORS


SYMBOL	MEANING
=	Equal to
<	Less than
<=	Less than or equal to
>	Greater than
>=	Greater than or equal to
<> or !=	Not equal to

Selecting Rows with Comparison Operators

Another Example:

```
SELECT *  
      FROM part  
      WHERE part_number =  
            'AX12';
```

Note criteria is in single quotes
PART_NUMBER is a character field



RESULT - Returns **all** fields but only the single record where the part_number field matches the given criteria

PART NUMBER	PART DESC	ON HAND	CLASS	WAREHOUSE	PRICE
AX12	Iron	104	HW	3	23.95

Sorting Output

Data is displayed in the order which it was added to the tables initially

- **Not always true** - some DBMS's will sort tables by their primary key (PK) automatically if no sorting criteria was specified.
 - Primary Keys are also automatically indexes (more on indexes later)
- You can specify the **direction of the sorting** by using **ASC** or **DESC**.
 - ASC is the default direction and is therefore optional
 - to sort data in descending order, use the **DESC** keyword after each field specified in the **ORDER BY** clause that is to be displayed in descending order

Sorting Output

To change the order the data is displayed in, use the `ORDER BY` clause in the `SELECT` statement:

```
SELECT *  
  FROM part  
 WHERE on_hand > 90  
 ORDER BY on_hand;
```

RESULT - Returns **all** fields but only those records where `on_hand > 90`

PART NUMBER	PART DESC	ON HAND	CLASS	WAREHOUSE	PRICE
BH22	Cornpopper	95	HW	3	24.95
AX12	Iron	104	HW	3	23.95
CX11	Blender	112	HW	3	22.95

Sorting Output – Multiple Columns

You can sort by more than one column

- The second column will ONLY be sorted If and Only If the first column has duplicates and then only those duplicates are sorted

```
SELECT *  
      FROM part  
      ORDER BY class,  
on_hand;
```

RESULT

PART_NUMBER	PART_DESC	ON_HAND	CLASS	WAREHOUSE	PRICE
BT04	Gas Grill	11	AP	2	150
BZ66	Washer	52	AP	3	400
CA14	Griddle	78	HW	3	39.99
BH22	Cornpopper	95	HW	3	24.95
AX12	Iron	104	HW	3	23.95
CX11	Blender	112	HW	3	22.95
AZ52	Dartboard	20	SG	2	12.95
BA74	Basketball	40	SG	1	29.95
CB03	Bike	44	SG	1	300
CZ81	Treadmill	68	SG	2	350

SQL Comparison Operators

WHERE

- In/ not in
 - Determines if the value of a single field is in a provided comma separated list
- Between/ not between
 - A range of data
- Like / not like
 - Activates the ability to use wildcards and patterns
- Is null /Is not null
 - Required fields vs. optional fields

SQL Comparison Operators

IN / NOT IN

- Determines if the value of a single field is in a provided comma separated list

SAMPLES

```
SELECT *  
  FROM part  
 WHERE class IN('HW', 'AP');
```

```
SELECT *  
  FROM part  
 WHERE warehouse IN(1, 2);
```

SQL Comparison Operators

BETWEEN / NOT BETWEEN

- Determines if the value of a single field is within a range provided
- Be careful with inclusive vs exclusive
 - Suggested to use some practise data to determine if the values given are included or excluded from the range

SAMPLES

```
SELECT *  
      FROM part  
      WHERE price BETWEEN 10 AND 50;
```

```
SELECT *  
      FROM part  
      WHERE part_number BETWEEN 'BA' AND  
      'CA';
```

SQL Comparison Operators

IS NULL / NOT IS NULL

- Determines if the value of a single field is NULL

SAMPLES

```
SELECT *  
    FROM part  
    WHERE price IS NULL;  
-- this returns prices that need to still be entered
```

```
SELECT *  
    FROM part  
    WHERE price IS NOT NULL;  
-- this returns only those products where the price  
has been entered
```

SQL Comparison Operators

LIKE / NOT LIKE

- Is a comparator that allows the use of wildcards

SAMPLES

```
SELECT *  
    FROM part  
    WHERE class LIKE 'HW';  
-- this is the same as class = 'HW' but now can use wildcards
```

```
SELECT *  
    FROM part  
    WHERE part_desc LIKE 'B%';  
-- this returns all products whose descriptions start with a capital B
```

Wildcards

String Wildcards

- Allow scripting when part of the required strings are unknown
 - Examples: starts with, ends with, contains
 - Use the % wildcard as a placeholder of unknown characters and unknown length

```
SELECT *  
      FROM part  
      WHERE part_desc LIKE 'B%';
```

-- this returns all products whose descriptions start with a capital B and any number of characters afterwards

Wildcards continued

```
SELECT *  
FROM part  
WHERE upper(part_desc) LIKE '%D';
```

Returns all rows where the part description ends with the letter 'D'.

- Note the use of the single function `upper()`. Strings are case sensitive in SQL and therefore we must control the statement as we can not assume the data in the database was entered in any specific way

Wildcards continued

```
SELECT *  
FROM part  
WHERE lower(part_desc) LIKE '%pop%';
```

Returns all rows where the part description contains the phrase 'POP' within it (at any location, including start, middle and end).

- Note the use of the single function `lower()`. Strings are case sensitive in SQL and therefore we must control the statement as we can not assume the data in the database was entered in any specific way.

CRUD

CRUD is a term most programmers should become familiar with

C - Create (**INSERT**)

R - Read (**SELECT**)

U - Update (**UPDATE**)

D - Delete (**DELETE**)

This term is used consistently throughout the database and programming industries.

These are DML statements.

INSERT

Inserting **NEW** records (rows) into a database

- There are a few different ways to do insert statements, for now we will cover just 2
 - We will ignore tables with automatically generated primary keys for now

Syntax 1

```
INSERT INTO <table name>  
    (<comma separated field list>  
VALUES  
    (<comma separated value list>); optional
```

- Order of fields and values must match
- All Required fields must be included

INSERT

Inserting **NEW** records (rows) into a database

Syntax 2

```
INSERT INTO <table name>  
VALUES  
(<comma separated value list>);
```

- In this syntax, the order and inclusion of fields are NOT optional
 - All fields must be included and
 - the values must be provided in the SAME order that the fields are in the table design

INSERT

Samples

```
INSERT INTO parts  
    (part_desc, part_number, on_hand, class,  
warehouse, price)  
VALUES  
    ('Soccer Ball', 'RC16', 26, 'SG', 1, 24.95);
```

```
INSERT INTO parts  
VALUES  
    ('RC16', 'Soccer Ball', 26, 'SG', 1, 24.95);
```

Note: the order of the fields

UPDATE

To update already **EXISTING** records (Rows)

- Only include those fields that are changing
- Make sure you have a WHERE clause that specifies EXACTLY what records are to be updated.

GENERIC SYNTAX

```
UPDATE <table name> SET  
    <field1>=<value1>,  
    <field2>=<value2>,  
    ....  
WHERE  
    <conditional statement>;
```

UPDATE continued

Samples

Updates a single record and changes the description

```
UPDATE parts SET  
    part_desc = 'Gas/Propane Grill'  
WHERE part_number = 'BT04';
```

Updates ALL records matching the WHERE criteria and increases the price by 10%

```
UPDATE parts SET  
    price = price * 1.10  
WHERE class = 'SG';
```

DELETE

Removes data from the table

- Most of the time, this is not easily undone (be very careful)
- 99% of the time a WHERE clause is used to specify the EXACT records to be deleted

GENERIC SYNTAX

```
DELETE FROM <table name>  
WHERE  
<conditional statement>;
```

DELETE continued

Samples

Deletes a single record from the parts table (because PK is specified)

```
DELETE FROM parts  
WHERE part_number = 'BT04';
```

Deletes ALL records matching the WHERE criteria - oops maybe

```
DELETE FROM parts  
WHERE warehouse = 2;
```


Thank you!