


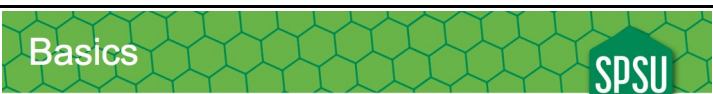
# SQL


## Brief Intro

**Orlando Karam**  
**okaram@spsu.edu**



SQL stands for Structured Query Language, and is the standard language for relational DBMSs. Sometimes it is pronounced 'sequel' and sometimes spelled, es qiu el. Every Relational DBMS implements SQL with some variations and extensions.







### Basics

- Structured Query Language
- Standard language for relational databases
- Every RDBMS implements, with slight variations, including
  - Special data types
  - Stored procedures
  - Kinds of indexes, file organization ...

2

SQL can be divided into three subsets, the Data Definition Language, or DDL, which allows us to create the database schema; the Data Manipulation Language, which allows us to manipulate the data in a database, and the Data Control Language, which allows us to manage users, permissions and such. In these slides, we give a brief introduction to the basic features of the DDL and the DML.





### Basics

- Data Definition Language (DDL)
  - CREATE / DROP / ALTER
    - Table
    - View
    - Index, Sequence, ...
- Data Manipulation Language (DML)
  - INSERT
  - UPDATE
  - DELETE
  - SELECT
  - Transactions
- Data Control Language (DCL)

3

Before we start, let me mention a few things you may see in the SQL code. First of all, comments can be done like in C, or using two dashes – to comment until the line ends.

SQL is case insensitive, except for data; however, it is customary to type reserved words using all capital letters, a convention that I try to follow.

Another thing that you need to keep in mind, is that SQL uses single quotes for character constants, instead of double quotes, which most other languages now use. This will probably cause you a little trouble until your fingers learn it :)

Finally, although not part of the statement, on text-based clients we use a semicolon to terminate a statement; the client will then send the statement to the server; if you're using a GUI client, or using SQL within a programming language, the semicolon is not required and may even be prohibited.

OK, so let's start by looking at how to create a table; surprisingly, we use a CREATE TABLE statement; we write CREATE TABLE, then the name of the table, and then a list of fields, between parenthesis.

For each field, we write the name, its data type, and then the constraints for the field, if any.

For example, in the code at the bottom, we are creating a table, called Book, with 3 fields, ISBN, Title and Pages; for ISBN we say it is a character or string type, nine characters long, and it is the primary key, which implies it is unique and null values are not allowed; title is a variable character type, up to 20 characters long, it is unique across the table, and null values are not allowed; finally we say pages is an integer field.

## Conventions

SPSU

- -- comments until end of line
- /\* can also use C-style comments \*/
- SQL is case insensitive (except for data)
- But we usually type reserved words in ALL CAPS
- Use single quotes for 'character constants'  
–programming quotes, don't let your word processor auto-correct them :) 'wrong'
- On text-based clients, end statement with semicolon ;

4

## CREATE TABLE

SPSU

```
CREATE TABLE table_name (  
    field type constraints,  
    field2 type2 ,  
    CONSTRAINT name ...,  
    ...  
);
```

```
CREATE TABLE Book (  
    ISBN CHAR(9) PRIMARY KEY,  
    Title VARCHAR(20) UNIQUE NOT NULL,  
    Pages Integer  
);
```

5

Although the data types supported may vary depending on the DBMS, some of the most used SQL datatypes include:


CHAR, which is a fixed length string type; if the data entered is smaller than the field length, spaces will be added at the end; strings larger than the field length could be truncated or, more commonly, not allowed by the DBMS.

Another common type is VARCHAR, which is for variable-length strings; smaller strings are not modified at all, and larger strings may be truncated or disallowed by the DBMS.

SQL caters heavily for business applications, so it has a fixed-precision numeric type, called NUMERIC. For this datatype we have two parameters, the first one is the *total* number of digits and the second one determines how many digits go after the decimal period. Notice most DBMSs also support traditional ints, of different sizes, and also several kinds of floating point types.

Finally, most DBMSs also support ways of represent dates and specific points in time; the DATE datatype contains only the data, and the TIMESTAMP datatype allows to specify a date and time. Most DBMSs will support several variations of these types.

## Common Datatypes



- **CHAR(n)**
  - fixed length strings, padded with spaces at end
- **VARCHAR(n)**
  - variable length strings, but no longer than n
- **NUMERIC(prec,dec)**
  - fixed precision numbers (not floats)
  - precision is **total** number of digits
  - dec is how many after the decimal point
  - NUMERIC(3,2) max value is 9.99
- **DATE, TIMESTAMP**
  - Represent dates, or specific points in time

6


Many times we want to constraint which data can go into the database, so *obviously wrong* data doesn't get in.

By default, NULL, a special value indicating that no data is present, is allowed in every field; if we want to disallow nulls, then we specify NOT NULL.

We also have UNIQUE, which requires each value for that field occurs at most once within the table.

For every table we should specify the PRIMARY KEY, which usually is just one field; PRIMARY KEY implies both UNIQUE and NOT NULL.

## Common constraints



- **NOT NULL**
- **UNIQUE**
- **PRIMARY KEY**
- **REFERENCES (foreign key)**
  - after REFERENCES put name of table, then field in parenthesis
  - StudentId REFERENCES Student(Id)
- **CHECK**
  - Allows for predicates after
  - CHECK(age>20)

7

Notice that NULL is different from 0 or the empty string; it is a special value signifying 'no data present', regardless of the reason; we may not have data because there's no value, we don't know the value, or a value for that row wouldn't make sense.

Another important constraint is REFERENCES, which establishes that a field is a foreign key, and which field from another table it references; after the word REFERENCES, we write the name of the table it references and then the name of the primary key of that table in parenthesis.

The last one we'll see is CHECK, which allows us to specify an arbitrary condition or predicate.

Now let's look at another example; we're creating a table named student, we have id, 3 characters long which is its primary key; then name, up to 20 characters and not null; then age, which is an integer, and we use CHECK to specify that it has to be at least 1 and at most 99; then we have Gender, being one character long.

Then we have Deg\_code, which is two characters long and references the field named code of the table named degree;

Finally we have Major and Credits.

## CREATE TABLE Example

SPSU

```
CREATE TABLE Student (
  Id CHAR(3) PRIMARY KEY,
  Name VARCHAR(20) NOT NULL,
  Age CHECK(Age>0 AND AGE<100),
  Gender CHAR NOT NULL,
  Deg_code CHAR(2) NOT NULL REFERENCES Degree(code),
  Major CHAR(3),
  credits INTEGER
);
```

8

After we've created a table, we probably want to insert some rows; we do that with the INSERT statement.

We write INSERT INTO, then the name of the table, then the fields we're providing data for in parenthesis, the keyword VALUES and finally the values, again in parenthesis. Remember that, when specifying the data, string constants use single quotes.

In the example statement, we are inserting a row into the student table; the value for the id field is 1, name is 'Orlando Karam, Major is 'CS' and Age is 21.

## INSERT

SPSU

- INSERT INTO table (fields) VALUES (values)
- Character constants have single quotes 'a'

```
INSERT INTO Student (Id,Name,Major,Age)
VALUES (1,'Orlando Karam','CS',21);
```

9

Now, after we have some data in, we may want to look at the data; we use the SELECT statement to get data from tables in a database.

The basic form of the SELECT statement is SELECT then the fields we want, then FROM and the table, and finally WHERE and the conditions that specify which rows we want. Notice if we use \* instead of the fields, we get all fields, and if we don't write WHERE we get all rows.

In the example on the left, we get all fields from the student table; in the example on the right, we get only the id and name, and only from those students who are majoring in CS.

## SELECT (Retrieving Data)

SPSU

- SELECT fields FROM table WHERE conditions
- Can use fields or expressions (a+3), \* for all fields
- Conditions use normal operators (=,>) and are combined with AND, OR, NOT

```
SELECT *
FROM Student

SELECT Id,Name
FROM Student
WHERE Major='CS'
```

10



The SELECT statements can be a little more complicated; first, we can use expressions instead of fields, and we get the result of that expression; SQL supports the usual arithmetic operations, and many other functions on numbers, strings and other datatypes; in our first example, we are getting the id, name and the rows age, with 5 added to it. Notice this is a temporary result, SELECT statements do NOT alter the table in any way; we are NOT adding 5 to the age *in* the table, just in our result.

Also, notice we can combine conditions, with the usual boolean operations; however, SQL uses the actual words AND, OR and NOT instead of weird symbols, so in the first example we get only rows for female CS majors.

In the second example we illustrate that we can alias or rename a field in the results; we're getting 5 plus the age, but we're calling it AgeIn5; we also show how to do an OR. Notice in both the conditions and the expressions, the usual precedence rules apply, or we can use parenthesis to modify which operations get evaluated first.

If we need to delete some rows from a table, we use the DELETE statement; we use the keywords DELETE FROM, then the name of the table, then a WHERE clause and the conditions specifying which rows we want to delete; if no conditions are specified, then all rows in the table will be deleted, but the table schema will still be in the database.

Notice we do not specify any fields for the DELETE statement, since we're always deleting the whole row.

## More Examples

SPSU

```
SELECT Id, Name, Age+5
FROM Student
WHERE Major='CS' AND Gender='F'
```

```
SELECT Id, Name, Age+5 AS AgeIn5
FROM Student
WHERE Age>=20 OR Age <=10
```

11

## DELETE

SPSU

- DELETE  
FROM table  
WHERE conditions
- If no conditions, all data
- Does NOT delete the meta-data, use DROP TABLE for that

```
DELETE
FROM Student
WHERE Id=1
```

12

If we need to change some rows in a table, we use the UPDATE statement; we use the keyword UPDATE, then the table name, then the keyword SET, then a field name, = and the value; if you want to set more than one field, separate with commas; finally, the keyword WHERE and then the conditions specifying which rows you want to update; as usual, if you don't specify a WHERE clause you modify all rows in the table.

Notice that in the SET clauses, we can use the field we're modifying in the expression, as in programming languages; for example, the last example on the slide adds 1 to the age for all rows of the student table, making each student one year older.

Now lets do another example. Imagine we want to represent students and their majors, with a relational schema diagram as in the slide. Try to write CREATE TABLE statements yourself. No peeking !

Here's one possible solution; your datatypes may have varied a little.

Notice that the table 'major' needs to be defined first, since the student table will reference it. Also, although the datatypes on your solution do not need to be identical, the datatype for the MajorId field of the Student table needs to match the Id field of the Major table.

## UPDATE (change data) SPSU

- UPDATE table  
SET field=value  
WHERE conditions

```
UPDATE Student
SET Name='Alfredo Karam', Age=25
WHERE Id=1
;
```

```
UPDATE Student
SET Age=Age+1
;
```

13

## Complete Example SPSU

- What would CREATE TABLE statements look like
  - Can invent reasonable data types (in real world, you keep a data dictionary when doing requirements/data modeling)

14

## Complete Example SPSU

```
CREATE TABLE Major(
  Id CHAR(3) PRIMARY KEY,
  Name VARCHAR(25) NOT NULL
);
```

```
CREATE TABLE Student (
  Id int PRIMARY KEY,
  Name VARCHAR(20) NOT NULL,
  MajorId CHAR(3) REFERENCES Major(Id)
);
```

- Major needs to be defined first
- The datatypes of Major.Id and Student.MajorId should match

15

Now let's look at insert statements. We are inserting one major, with id 'CS' and named 'Computer Science'; we then insert a student with id 1, named Orlando, and majoring in 'CS'; notice we use CS for the majorId, not 'Computer Science', since we're storing a reference to the major table, its id.

## Now insert statements

SPSU

```
INSERT INTO Major (Id,Name) VALUES
('CS','Computer Science');
INSERT INTO Student (Id,Name,MajorId) VALUES
(1, 'Orlando', 'CS');
```

- Notice field names and values match
- Notice value in Student.MajorId needs to match a value of Major.Id

16

Now you try ... insert majors for IT and SWE and students majoring there. No peeking ...

## Now you try

SPSU

- Insert majors for IT and SWE
- Insert students majoring there

17

And here's one possible solution; again, your solution may vary, since you'll choose different values; however, a few things need to match; first, the student ids need to be different, since id is the primary key for the student table; also, the majorId values for the student rows you create need to be the values SWE and IT.

## Now you try - Solution

SPSU

- Insert majors for IT and SWE
- Insert students majoring there

```
INSERT INTO Major (Id,Name) VALUES
('SWE','Software Engineering');
INSERT INTO Student (Id,Name,MajorId) VALUES
(2, 'Frank', 'SWE');

INSERT INTO Major (Id,Name) VALUES
('IT','Information Technology');
INSERT INTO Student (Id,Name,MajorId) VALUES
(3, 'Andy', 'IT');
```

18

Now lets do a simple select. To show all the students, you'd do `SELECT * FROM Student`; now you try to do all the majors ... and then the id and name of students majoring in CS ... no peeking !

## Basic SELECTS

SPSU

- Show all the students

```
SELECT *  
FROM Student;
```

- Now you do all majors

- And then Id and name of students majoring in CS

19

To display all majors, we just do '`SELECT * from Major`'; and to get the id and name of students majoring in CS we do `SELECT Id,Name, from Student`, and then add a `WHERE` clause, `WHERE Major='CS'` ; notice we're using the major's id, CS, and that we're restricting ourselves to querying one table; we'll later see how to do queries that join more than one table.

## Basic SELECTS

SPSU

- Now you do all majors

```
SELECT *  
FROM Major;
```

- And then Id and name of students majoring in CS

```
SELECT Id, Name  
FROM Student  
WHERE MajorId='CS'  
;
```

20

Now let's look at `UPDATE` statements; if we want to set the major of the student with id 2 to be IT (in our example, it was Frank and an SWE major) then we would do: `UPDATE student, SET Major='IT', WHERE Id=2`.

Now you try it ... change the name of the student with id 3 to be Rich

## Updates

SPSU

- Change the student with id 2 (Frank) to be IT instead of SWE

```
UPDATE Student  
SET MajorId='IT'  
WHERE Id=2  
;
```


- Now you do - change the name for student with id 3 to be Rich

21



Ok, it's easy, right ? Just UPDATE student,  
SET Name='Rich' WHERE Id=3

## Updates



- Now you do - change the name for student with id 3 to be Rich


```
UPDATE Student
SET Name='Rich'
WHERE Id=3
;
```

22

Now DELETE; to delete all students  
majoring in IT we do ... DELETE FROM  
Student WHERE MajorId='IT';

Now you try ... delete the IT major from the  
majors table

## DELETE



- Delete any students majoring in IT

```
DELETE
FROM Student
WHERE MajorId='IT'
;
```


- Now you do - Delete the IT major (from the majors table)

23

And here's the answer; easy, right ? DELETE  
FROM Major WHERE Id='IT'.

Now you know the basics of SQL; however,  
you need to get the syntax and semantics to  
become natural; the only way to do this, is to  
practice a lot. Get on the computer and try  
some examples; the more you practice, the  
better you get.

## DELETE



- Now you do - Delete the IT major (from the majors table)

```
DELETE
FROM Major
WHERE Id='IT'
;
```

24