Data Definition Language

SQL Syntax

SQL

- The standard programming language for creating, updating and retrieving information that is stored in databases
- It is:
  - A ***programming language*** – a formal language in which to write programs to create, modify, and query databases.
    - Defined by rules of ***syntax*** (determine the words and symbols you can use and how they are combined)

SQL

- It is ***declarative*** – you describe what you want and the database will determine how to do it
- It is ***interactive*** – you issue SQL commands directly to you Database Management System (Access, SQL Server, MySQL, etc) and it displays the results
- OR it is ***embedded*** – you can embed the SQL statements in a scripting language (PHP)

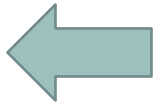
SQL

- It is ***standardized*** – no one owns it, but it is a standard that is defined by an international standards working group.
- It is commonly referred to as **S**tructured **Q**uery **L**anguage – but that is incorrect. It actually stands for just SQL

Syntax Conventions

- Each SQL statement begins on a new line
- The indentation level is two spaces
- Each clause begins on a new, indented line
- SQL is case insensitive, myname, MyName, and MYNAME are considered to be identical identifiers
- Generally use uppercase for SQL keywords and lowercase for user-defined values.

```
SELECT au_fname, au_lname
  FROM authors
  ORDER BY au_lname
```



You WILL follow these conventions

SQL Syntax

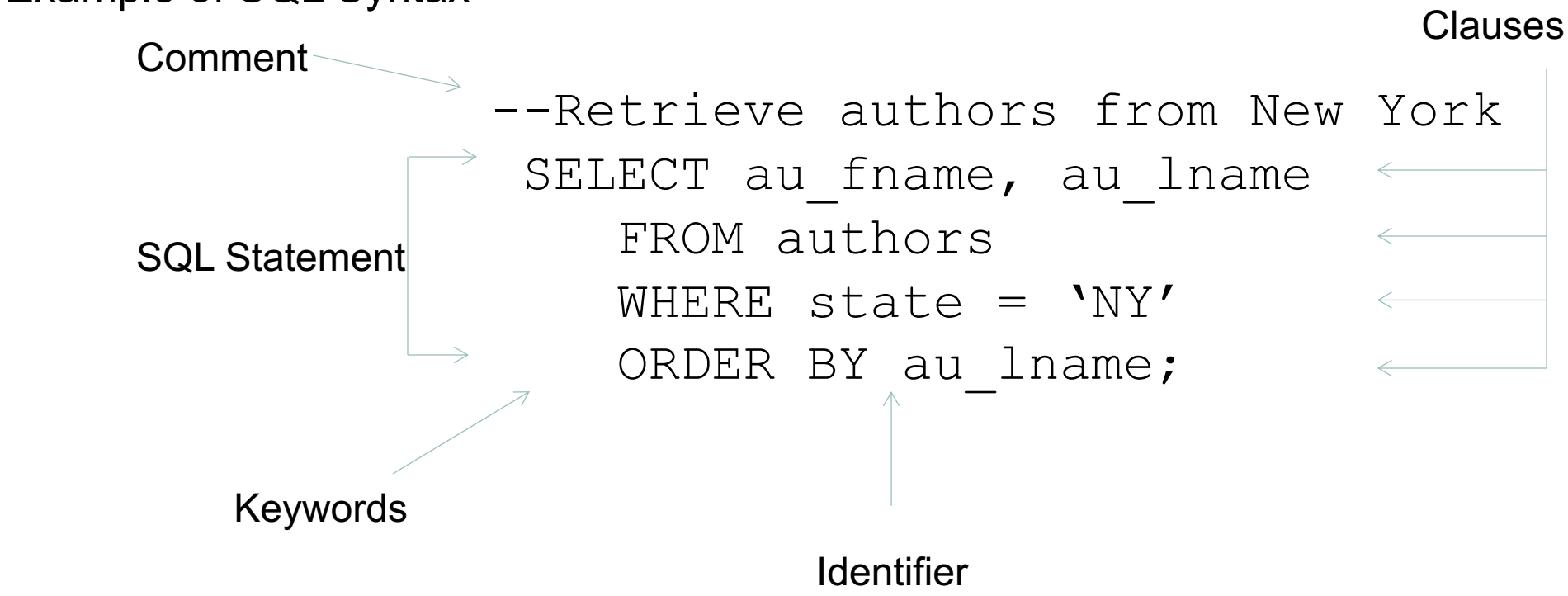
- Comment – text that explains your program
- SQL statement – a valid combination of **tokens** introduced by a **keyword.**
- Tokens – the indivisible particles of the SQL language. They include keywords, identifiers, operators, literals (constants) and punctuation symbols
- Clauses – a fragment of an SQL statement that's introduced by a keyword, is required or optional and must be given in a particular order. (SELECT, FROM , WHERE and ORDER BY introduce the four clauses in the example)

SQL Syntax

- Keywords – words that SQL reserves because they have special meaning in the language.
  - SELECT, CREATE, TABLE
- Identifiers – words that you use to name objects, columns, aliases, indexes and views
  - authors, titles, books,
- Terminating semicolon – ends with an SQL statement (ACCESS and SQL server do not require)

SQL Syntax

Example of SQL Syntax



SQL Syntax

- Comment – text that explains your program
- SQL statement – a valid combination of **tokens** introduced by a **keyword.**
- Tokens – the indivisible particles of the SQL language. They include keywords, identifiers, operators, literals (constants) and punctuation symbols
- Clauses – a fragment of an SQL statement that's introduced by a keyword, is required or optional and must b given in a particular order. (SELECT, FROM , WHERE and ORDER BY introduce the four clauses in the example)

SQL Syntax

- Keywords – words that SQL reserves because they have special meaning in the language.
- Identifiers – words that you use to name objects, columns, aliases, indexes and views
- Terminating semicolon – ends with an SQL statement (ACCESS and SQL server do not require)

SQL Syntax

- SQL is a free-form language whose statements can :
  - Be in uppercase or lowercase - SELECT, Select, select, SeLeCt are considered to be identical
  - Continue on the next line
  - Be on the same line as other statements
  - Start in any column

Equivalent Statements

Statements are equivalent

```
SELECT au_fname,
au_lname
    FROM authors
    WHERE state = 'NY'
    ORDER BY au_lname;
```

```
Select au_fname
,             AU_LNAME
       FROM
Authors     WhERe   state
= 'NY' order
                                by
AU_lnamE
;
```

Unique Identifiers

Unique Identifiers

- Unique identifiers are used to generate primary-key values to identify rows
- Can be unique universally or within a specific table (serial number)
- The SQL standard calls columns with auto-incrementing values identity columns.
- MySQL – `auto_increment` attribute

How do we create tables

Data Definition Language

- Database objects like tables and columns are created, modified and removed from the database using DDL.
- Accomplished using the CREATE, ALTER and DROP statements

Table/Column Names

- Must be unique to each database
- Must meet the following rules:
  - Can be up to 128 characters long (MySQL – 64 characters long)
  - Must begin with a letter
  - Can contain letters, digits and underscores (_)
  - Can NOT contain spaces or special characters (such as #, $, &, % or punctuation)
  - Can’t be reserved keywords (for example you can’t call a table Select or Sum)

Table/Column Names

- You can use a quoted identifier (delimited identifier) to break some of the rules – this involves surrounding the name with double quotes or square brackets

```
[customer addresses]

"customer addresses"
```

It is not recommended

Naming recommendations

- Use lowercase letters
- names_with_underscores are easier to read than nameswithoutthem.
- Use the same conventions/abbreviations throughout the database

Column Data Types

- The data type of each column is a character, numeric, datetime or other data type
- Character String Types – has these characteristics:
  - Ordered sequence of 0 or more characters
  - Length can be varying or fixed
  - Is case sensitive (in that A comes before a in sorting)
  - Is surrounded by single quotes in SQL statements

Strings

- Length of the string is an integer between 0 and length.
- A string with no characters is called an empty string
- An empty string is considered to be a VARCHAR of length zero
- Keep columns as short as possible, rather than giving them room to grow in the future – shorter columns sort and group faster than longer ones

Creating Tables

Creating Tables

- To create a table it is necessary to name the table and all the attributes that compose it
- To create a table, you specify the following:
  - Table name
  - Column names
  - Data Types of the Columns
  - Default Values of columns
  - Constraints

Example

```
CREATE TABLE table
(
column1 data_type1 [col_constraint],
column2 data_type2 [col_constratint],
….

);
```

CREATE some tables

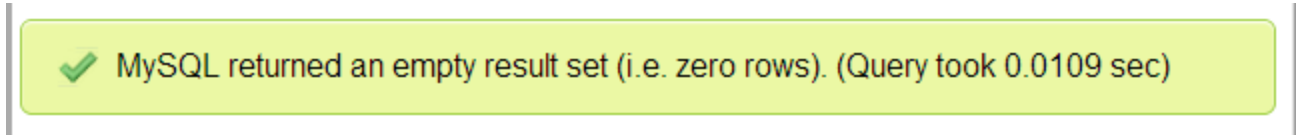
```
CREATE TABLE title
(
  title_id CHAR(3),
  title_name VARCHAR(40),
  type VARCHAR(10),
  pub_id CHAR(3),
  pages INTEGER,
  price DECIMAL(5,2),
  sales INTEGER,
  pubdate DATE,
  contract SMALLINT
);
```

```
CREATE TABLE title_authors
(
  title_id CHAR(3),
  au_id CHAR(3),
  au_order SMALLINT,
  royalty_share DECIMAL(5,2)
);
```

Table info

- If you try to create a table with a name that already exists you will get an error.

- A newly created table is empty – has 0 rows



Constraints

CONSTRAINTS

- Constraints let you define rules for values allowed in columns
- A column constraint is part of a column definition and imposes a condition on that column only
- A table constraint is declared independently of a column definition and can impose a condition on more than on column in a table.

Constraints

- NOT NULL – prevents nulls from being inserted into a column
- PRIMARY KEY – sets the primary key
- FOREIGN KEY – sets the foreign key
- UNIQUE – prevents duplicate values from being inserted into a column
- CHECK – limits the values that can be inserted into a column by using logical expressions

Constraints

- Assigning names to constraints lets you manage them efficiently
- It allows you to change or delete them later if required
- It is not uncommon to leave a NOT NULL constraint but you should name others
  - If you do not name your constraint, the DBMS will do it for you
  - System assigned names often contain strings of random characters and are cumbersome to use
  - The names also appear in warning, error messages and logs

Constraints

- To name a constraint use the following:

CONSTRAINT *constraint_name*

Where constraint_name is the name of the constraint – the names **must** be unique within a table

Working with Nulls

Forbidding Nulls

- A column's nullability determines whether is can contain nulls or in other words – whether it requires values or not
- Remember, a null:
  - Is not a value, but a marker that means no value has been entered
  - Represents a missing, unknown or inapplicable value
  - Is not the same as zero (0), a blank or an empty string
  - The keyword NULL represents a null
  - Belongs to no data type and be inserted into any column that allows null

Defining NULLs

- You define a null-ability constraint by using the keywords NOT NULL in the CREATE TABLE column definition
- Avoid allowing nulls because they complicate queries, insertions and updates
- Forbidding NULLS helps maintain data integrity – ensures data is entered
- If you don’t specify a NOT NULL, the column accepts NULLs by default

NOT NULL

```
CREATE TABLE authors
(
  au_id CHAR(3)           NOT NULL,
  au_fname VARCHAR(15)    NOT NULL,
  au_lname VARCHAR(15)    NOT NULL,
  phone VARCHAR(12),
  address VARCHAR(20),
  city VARCHAR(15),
  state CHAR(2),
  zip CHAR(5)
);
```

Working with Default Values

DEFAULT

- A default value is a value that is assigned to a column if you omit a value for the column when adding values to a table
- A default :
  - Applies to a single column
  - Is defined by using the keyword DEFAULT in a CREATE TABLE column definition
  - Must have the same data type as its column
  - Must fit in the column
  - If no default is provided, and the column is declared NOT NULL, then the system will refuse to add any information OR insert a 0 value

Examples

```
CREATE TABLE titles
(
  title_id          CHAR(3)       NOT NULL      ,
  title_name  VARCHAR(40)   NOT NULL      DEFAULT ‘’,
  type              VARCHAR(10)                 DEFAULT 'unknown',
  pub_id            CHAR(3)               NOT NULL,
  pages             INTEGER                             DEFAULT NULL,
  price             DECIMAL(5,2)  NOT NULL      DEFAULT 0.00,
  sales             INTEGER,
  pubdate     DATE                                        DEFAULT Date(),
  contract    SMALLINT              NOT NULL      DEFAULT 0
);
```

** in MySQL – Can’t set a default to a function – substitute with DEFAULT CURRENT_DATE

Verify defaults

Run this query ->

```
SELECT * FROM titles;
```

Then run this query ->

```
INSERT INTO titles (title_id, pub_id)
  VALUES ('T14','P01');
```

Finally run this query ->

```
SELECT * FROM titles;
```

Primary Keys

PRIMARY KEYS

Primary Keys –

- Identifies each row uniquely in a table
- No two rows can have the same primary-key value
- Primary keys do not allow NULLS
- Each table has exactly one primary key

Defining Primary Keys

- Define a primary-key constraint by using the keywords PRIMARY KEY
- No more than one primary-key constraint is allowed in a table
- Must be set to NOT NULL (if you fail to set it, it is implicitly set by the DBMS)
- Primary Key values normally don’t change after they’re changed

PRIMARY KEY Example

```
CREATE TABLE publishers
(
pub_id CHAR(3) PRIMARY KEY,
pub_name VARCHAR(20) NOT NULL,
city VARCHAR(15)   NOT NULL,
state CHAR(2),
country VARCHAR(15)     NOT NULL
);
```

PRIMARY KEY Example unnamed table constraint

```
CREATE TABLE publishers1
(
pub_id CHAR(3)NOT NULL,
pub_name VARCHAR(20)NOT NULL,
city VARCHAR(15)  NOT NULL,
state CHAR(2),
country VARCHAR(15)NOT NULL,
PRIMARY KEY (pub_id)
);
```

PRIMARY KEY – Example - Named table constraint

```
CREATE TABLE publishers2
(
pub_id CHAR(3) NOT NULL,
pub_name VARCHAR(20)NOT NULL,
city VARCHAR(15) NOT NULL,
state CHAR(2),
country VARCHAR(15)NOT NULL,
CONSTRAINT publishers_pk
PRIMARY KEY (pub_id)
);
```

Foreign Keys

Foreign Key

- A foreign key is a mechanism that associates two tables
- It's a column in a table whose values relate to or references values in some other table
- A foreign key ensures that rows in one table have corresponding rows in another table
- Establishes a direct relationship to a primary key in the referenced table
- Can allow nulls
- Are generally unique in their own table

Foreign Keys

- Define a foreign-key constraint using the keywords FOREIGN KEY or REFERENCES
- The data types must have the same data type

A FOREIGN KEY in one table points to a PRIMARY KEY in another table

FOREIGN KEY – NOT NAMED

```
CREATE TABLE royalties(
title_id CHAR(3) NOT NULL  PRIMARY KEY,
title_name VARCHAR(40) NOT NULL,
type VARCHAR(10),
pub_id CHAR(3) NOT NULL,
pages INTEGER,
price DECIMAL(5,2),
sales INTEGER,
pubdate DATE,
contract SMALLINT NOT NULL,
CONSTRAINT royalties_publishers_fk
FOREIGN KEY (pub_id)
REFERENCES publishers(pub_id)
);
```

Name is made up of two table names and fk for foreign key

This column

Other column

Other table

Foreign Key example

```
CREATE TABLE royalties
(
title_id CHAR(3) NOT NULL ,
advance DECIMAL (9,2),
royalty_rate DECIMAL(5,2),
CONSTRAINT royalties_pk
  PRIMARY KEY (title_id),
CONSTRAINT royalties_title_id_fk
  FOREIGN KEY (title_id)
  REFERENCES titles(title_id)
);
```

Alias

table

column

Unique Constraints

UNIQUE

- A unique constraint ensures that a column(s) contain(s) no duplicate values.
- Similar to a primary-key constraint except that they can contain NULLS & a table can have multiple unique columns

UNIQUE

- Define a unique constraint by using the keyword UNIQUE
- A table can have zero or more unique constraints
- A PRIMARY KEY constraint automatically has a UNIQUE constraint defined on it.

UNIQUE example

```
CREATE TABLE titles
(
title_id CHAR(3) NOT NULL PRIMARY KEY,
title_name VARCHAR (40) NOT NULL UNIQUE ,
type VARCHAR(10),
pub_id CHAR(3) NOT NULL,
pages INTEGER,
price DECIMAL (5,2),
sales INTEGER ,
pubdate DATE,
contract SMALLINT NOT NULL,
);
```

UNIQUE example

```
CREATE TABLE titles(
title_id CHAR(3) NOT NULL,
title_name VARCHAR (40) NOT NULL,
type VARCHAR(10),
pub_id CHAR(3) NOT NULL,
pages INTEGER,
price DECIMAL (5,2),
sales INTEGER ,
pubdate DATE,
contract SMALLINT NOT NULL,
CONSTRAINT titles_pk
  PRIMARY KEY (title_id),
CONSTRAINT titles_unique1
  UNIQUE (title_name)
);
```

Check Constraint

CHECK Constraint

- Check constraints further limit the values that a column or set of columns accept.
- Commonly check the following:
  - Min/Max values
  - Specific values
  - Range of values

CHECK (salary <=50000)

Would check to make sure that no salary exceeds 50,000

CHECK Constraint

- Define a check constraint by using the keyword CHECK
- A column can have zero or more check constraints associated with it
- The constraint's condition is almost any valid WHERE condition
  - Comparison (=,<>,<,<=,>,>=), LIKE, BETWEEN, IN or IS NULL
  - Join multiple conditions with AND, OR and NOT
  - Can refer to any column in the table, but can't refer to columns in other tables

CHECK Example

```
CREATE TABLE cartitems
(
cart_id INTEGER NOT NULL,
item_id INTEGER NOT NULL,
qty SMALLINT NOT NULL CHECK (qty <=10)
);
```

NOTE – this will 'work' in MySQL – it will create the table, but does not enforce the constraint

Creating Temporary Tables

TEMPORARY TABLES

- Commonly used to:
  - Store the result of a complex, time-consuming query once and use the result in subsequent queries
  - Create an image or snapshot at a particular moment in time
  - Hold the result of a subquery
  - Hold intermediate results of long or complex calculations

Temporary Tables

- The temporary table is emptied automatically at the end of the session (time you are connected to the DBMS) or transaction
- Temporary tables follow the same rules with regard to names, column names, data types, etc.
- Define a temp table with CREATE TABLE and then additional syntax:
  - GLOBAL TEMPORARY (available to you and other users)
  - LOCAL TEMPORARY (available only to you)

Temporary Table Example

- Temporary table has no rows initially

```
CREATE LOCAL TEMPORARY TABLE editors
(
  ed_id CHAR(3),
  ed_fname VARCHAR(15),
  ed_lname VARCHAR(15),
  phone VARCHAR(12),
  pu_id CHAR(3)
);
```

Creating Copies of Tables

CREATE TABLE AS

- Create a copy of a table to:
  - Archive specific rows
  - Make backup copies of tables
  - Create a snapshot of a table at a particular moment in time
  - Duplicate a table's structure but not its data
  - Create test data or to test INSERT, UPDATE or DELTE operations before modifying production data

CREATE TABLE …. AS SELECT

```
CREATE TABLE authors2 AS
 SELECT * FROM authors
[WHERE……..]
```

OR SQL and ACCESS

```
SELECT *
INTO authors2
FROM authors
[WHERE……]
```

To copy just the structure and not the data, use a where clause that returns no records

Altering Tables

ALTER TABLES

- Use ALTER TABLE to modify a table definition by adding, altering or dropping columns and constraints
- Use alter to:
  - Add of drop a column
  - Alter a column's data type
  - Add, alter, or drop a column's default value or nullability constraint
  - Rename a column
  - Rename a table

ALTER a table

ALTER TABLE *table*

alter_table_action

- Where table is the name of the *table* that is being altered
- Where alter_table_action can be and begins with the keyword ADD, ALTER or DROP

ADD COLUMN column type [constraints]

```
ALTER TABLE authors
  ADD COLUMN email_address VARCHAR(25)
```

ALTER a table

```
ALTER TABLE publishers
ALTER COLUMN city SET DEFAULT 'Windsor'


ALTER TABLE authors
DROP COLUMN email_address
```

You can't drop the table's only remaining column – in order to delete the table you must use the DROP keyword

Deleting Tables

DROP TABLES

- Use DROP TABLE to remove a table from a database
  - You can drop a base table or a temporary table
  - Dropping a table destroys its structure, data, indexes, constraints, permissions and so on.
  - Dropping a table IS NOT the same as deleting all its rows. You can empty a table of rows and NOT destroy it.

Drop Tables

- Dropping a table does not drop views that reference it so you will have to make adjustments or drop them as well (same with foreign keys)
  - There are specifications you can make for drop behavior – RESTRICT will not allow you to drop a table that is referenced by views or other constraints. CASCADE causes referencing objects to be dropped along with the table.

- `DROP TABLE titles;`

- `DROP TABLE authors2;`

Show Tables and Table Structures

Show Tables

- In order to see all the tables that currently exist in your database, use the SHOW TABLES command

Tables_in_dtakakidb
authors

```
1 SHOW TABLES;
```

- It will display a list of all the tables that have been created.

Show Table Structure

- To see the structure of any table in your database, use the DESCRIBE command followed by the table's name
- For example to see the structure of the authors table:

Field	Type	Null	Key	Default	Extra
au_id	char(3)	NO			
au_fname	varchar(15)	NO			
au_lname	varchar(15)	NO			
phone	varchar(12)	YES			
address	varchar(20)	YES			
city	varchar(15)	YES			
state	char(2)	YES			
zip	char(5)	YES			

```
DESCRIBE authors;
```