MITx | MicroMasters™
in Supply Chain Management

MIT Center for Transportation & Logistics

CTL.SC4x – Technology and Systems

Key Concepts Document

This document contains the Key Concepts for the SC4x course, Weeks 3.

These are meant to complement, not replace, the lesson videos and slides. They are intended to be references for you to use going forward and are based on the assumption that you have learned the concepts and completed the practice problems.

The first draft was created by Dr. Alexis Bateman in the Winter of 2017.

This is a draft of the material, so please post any suggestions, corrections, or recommendations to the Discussion Forum under the topic thread "Key Concept Documents Improvements."

Thanks,

Chris Caplice, Eva Ponce and the SC4x Teaching Community
Winter 2017 v1

This work is licensed under a Creative Commons Attribution-NonCommercial-ShareAlike 4.0 International License.

Database Queries

Summary

As we continue our discussion of database management, we dive into the issue of database queries. The ability to make effective queries in a large database enables us to harness the power of big data sets. SQL (Structure Query Language) is a language and contains the commands we use to create, manage, and query relational databases. As in all technology and systems applications, there are a multitude of vendors who offer SQL variations, but in general they have a common set of data types and commands. SQL is portable across operating systems and in general, portable among vendors. Having covered the commonly used data types in previous lessons, in this next section we will cover very commonly used queries.

Structured Query Language

SQL is used to query, insert, update, and modify data. Unlike Java, Visual Basic, or C++, SQL is not a complete programming language; it is a sub-language of approximately 30 statement types. It is generally embedded in another language or tool to enable database access. A few definitions we need to be aware of as explore SQL are:

- **Data definition:** Operations to build tables and views (virtual tables)
- **Data manipulation:** INSERT, DELETE, UPDATE or retrieve (SELECT) data
- **Data integrity:** Referential integrity and transactions n Enforces primary and foreign keys
- **Access control:** Security for multiple types of users
- **Data sharing:** Database accessed by concurrent users

A few issues to make note as you work with SQL is that it has several inconsistencies. For example, NULLs can be problematic and we will explore that later. In addition, when working with SQL it is important to recognize that it operations on declarative language, not procedural language. This means that when you write the command in such a way that describes what you want to do, and not HOW you want to do it. It is left up to the application to figure it out.

Variations among SQL Implementation

Because different databases use SQL, there can be variation in how SQL is implemented. The variations include:

- Error codes
- Data types supported (dates/times, currency, string/text variations)
- Whether case matters (upper, lower case)
- System tables (the structure of the database itself)
- Programming interface (no vendor follows the standard)
- Report and query generation tools
- Implementer-defined variations within the standard
- Database initialization, opening and connection

This work is licensed under a Creative Commons Attribution-NonCommercial-ShareAlike 4.0 International License.

As we have already learned, a data type defines what kind of value a column can contain. However, because there is variation across databases – we will use the data types for MySQL for the for the purpose of this discussion. MySQL has three main data types: numeric, strings (text), and dates/times. See the following figures:

Core My SQL Data Types - Numeric

Numeric Type	Description
INT	A standard integer
BIGINT	A large integer
DECIMAL	A fixed-point number
FLOAT	A single-precision, floating-point number
DOUBLE	A double-precision, floating-point number
BIT	A bit field

Core MySQL Data Types – Strings (Text)

String Type	Description
CHAR	A fixed-length, non-binary string (character)
VARCHAR	A variable-length, non-binary string
NCHAR	Same as above + Unicode Support
NVARCHAR	Same as above + Unicode Support
BINARY	A fixed-length, binary string
VARBINARY	A variable-length, binary string
TINYBLOB	A very small BLOB (binary large object)
BLOB	A small BLOB
TEXT	A small, non-binary string

This work is licensed under a Creative Commons Attribution-NonCommercial-ShareAlike 4.0 International License.

Core MySQL Data Types – Dates/Times

Date / Time Type	Description
DATE	A date value in 'CCYY-MM-DD' format
TIME	A time value in 'hh:mm:ss' format
DATETIME	Date/Time in 'CCYY-MM-DD hh:mm:ss' format
TIMESTAMP	Timestamp in 'CCYY-MM-DD hh:mm:ss' format
YEAR	A year value in CCYY or YY format

Creating Databases and Tables

To get started, we will need to know how to create databases and tables. While MySQL can be an intimidating program, but once you master some of the basics, you will be able to work effectively with large data sets.

- To create a Database, we use the CREATE DATABASE command
- Once you have created the database, you will now apply the USE command to tell the system which database to use

Once you have created a database, you will want to create tables within the larger database:

- New tables are declared using the CREATE TABLE command
- We can also set the name and data type of each attribute
- When creating new tables, we can specify primary keys and foreign key relationships
- We can also decide whether or not NULL or empty values are allowed

Inserting Data into a new Database

Once you have created a new database, you are ready to insert data. The data model will act a guide to load data into a new database. If the database builds well, it may mean that you have found the real business rules. Or, if you have some errors, you may have the real business rules, but the data may be messy. Finally, if it builds with many errors – this may be the case that the business rules are not accurate and are closer to what they want or think they have, not what they use. In many cases, it is useful to get sample data and browse it during the process of building the model.

SQL Select Queries

SQL SELECT query is used to fetch the data from a database table, which returns data in the form of a result table. SELECT returns a set of attributes in a query. In most applications, SELECT is the most commonly used data manipulation language command. SELECT statements are constructed from a series clauses to get records from one or more tables or views.

Clauses must be in order; only SELECT and FROM are required:

- SELECT *attributes/columns*

This work is licensed under a Creative Commons Attribution-NonCommercial-ShareAlike 4.0 International License.

- INTO *new table*
- FROM *table or view*
- WHERE *specific records or a join is created*
- GROUP BY *grouping conditions (attributes)*
- HAVING *group-property (specific records)*
- ORDER BY *ordering criterion* ASC | DESC
- DISTINC return *distinct values*

Wildcards in SQL

Most database implementations offer additional regular expressions – wildcards. A wildcard character can be used to substitute for any other character(s) in a string. Regular expressions can be used to find records, which match complex string patterns. For instant, MySQL has:

- [list] match any single character in list, e.g. [a-f]
- [^list] match any single character not in list, e.g. [^h-m]

Editing a Table

In some cases you will be faced with the need to edit a table. In this case you will use the following:

- INSERT is used to add a new record to a table that contains specific values for a set of attributes in that table
- The UPDATE keyword is used to modify a specific value or set of values for a set of records in a table
- DELETE is used to remove records from a table that meet a specific condition

Learning Objectives

- Become more familiar with SQL.
- Recognize different implementations of SQL have differences to be aware of.
- Review the different data types.
- Learn how to create new databases and tables.
- Understand how to SELECT query.
- Be familiar with wildcards and when to use them.
- Review how to edit a table.

This work is licensed under a Creative Commons Attribution-NonCommercial-ShareAlike 4.0 International License.

Database Conditional, Grouping, and Joins

Summary

In the next section we examine how to deal with database conditional, grouping, and joins. As we get further into SQL, we will need to refine our approach to make our actions more effective. For example, we will need to narrow the set of records that get returned from a query. We will also need to make statistical queries across different groupings or records. In addition, we will need to sample or order our output results. Another challenge will include integrating data from other sources within our database. The following review will cover these challenges and others as we continue to work with SQL.

Database Conditional Clauses

A conditional clause is a part of a query that restricts rows matched by certain conditions. You can narrow SELECT statements with conditional clauses such as WHERE IN, or the BETWEEN keyword. WHERE IN statements are used to identify records in a table with an attribute matching a value from a specified set of values. The BETWEEN keywords are used to identify records that have values for a particular attribute that fall within a specified range

WHERE IN: WHERE *attribute* IN is used to select rows that are satisfied by a set of WHERE conditions on the same attribute

BETWEEN Keyword: Select records where the attribute value is between two numbers using BETWEEN, range is inclusive and also works with time and date data.

Null Values

Null values are treated differently from other values; they are used as a placeholder for unknown or inapplicable values. If values are empty of missing, they are stored as NULL. A few issues to be aware of for NULL values:

- NULL values evaluate to NOT TRUE in all cases
- Check for NULLS using IS and IS NOT
- When a specific attribute may contain NULL or missing values, special care must be taken when using conditional clauses

This work is licensed under a Creative Commons Attribution-NonCommercial-ShareAlike 4.0 International License.

Grouping Data and Statistical Functions

Once you are a bit more comfortable working with SQL, you can start to explore some of the statistical functions that are included in many implementations of SQL. These functions can operate on a group of records. Using the GROUP BY clause will return a single value for each group of records. To further restrict the output of the GROUP BY clause to results with certain conditions, use the HAVING key words (analogous to the WHERE clause).

Aggregate Statistical Functions in SQL

Commonly used functions include:

Name	Description
AVG()	Return the average value of the argument
COUNT()	Return a count of the number of rows returned
COUNT(DISTINCT)	Return the count of a number of different values
MAX()	Return the maximum value
MIN()	Return the minimum value
STD()	Return the population standard deviation
STDDEV_SAMP()	Return the sample standard deviation
SUM()	Return the sum
VAR_POP()	Return the population standard variance
VAR_SAMP()	Return the sample variance
VARIANCE()	Return the population standard variance

More advanced statistical functions can be created using the basic statistical functions built into SQL such as calculating the weighted average or getting the z-score values by combining different functions.

Sorting and Sampling Data

You will also be faced with the need to sort and sample the data. Several clauses will help you will that including ORDER BY, LIMIT, and RAND.

ORDER BY: The ORDER BY clause specifies that the results from a query should be returned in ascending or descending order

LIMIT the number of returned records: A LIMIT clause restricts the number of records that would be returned to a subset, which can be convenient for inspection or efficiency

This work is licensed under a Creative Commons Attribution-NonCommercial-ShareAlike 4.0 International License.

Randomly select and order records: The RAND() function can be used to generate random values in the output or to randomly sample or randomly order the results of a query. For instance:

Reorder the entire table:
```
SELECT *
FROM table
ORDER BY RAND();
```

Randomly select a single record:
```
SELECT *
FROM table
ORDER BY RAND()
LIMIT 1;
```

Generate a random number in the output results:
```
SELECT id, RAND()
FROM table;
```

Creating New Tables and Aliases

AS Keyword (Aliases): The AS keyword creates an alias for an attribute or result of a function that is returned in a query

CREATE TABLE AS: Use CREATE TABLE with AS to create a new table in the database using a select query. It matches columns and data types based on the results in the select statement.

Results from a query can be inserted into a new table using the CREATE TABLE with the AS keyword. As seen in the following:

```
CREATE TABLE new_table
AS (    SELECT column_name(s)
        FROM old_table);
```

SELECT INTO: Results from a query can be inserted into an existing table using a SELECT INTO clause if the table with the appropriate structure already exists.Take the results of a select statement and put them in an existing table or database:

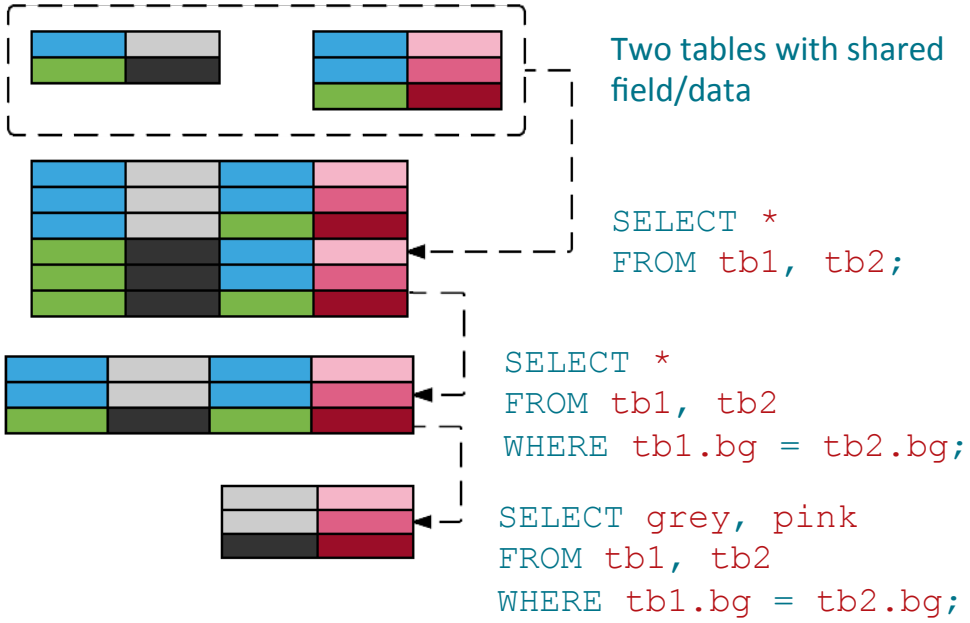
```
SELECT column_name(s)
INTO newtable [IN externaldb]
FROM table1;
```


This work is licensed under a Creative Commons Attribution-NonCommercial-ShareAlike 4.0 International License.

Joining Multiple Tables

The relational database model allows us join multiple tables to build new and unanticipated relationships. The columns in a join must be of matching types and also must represent the same concept in two different tables. This can help us to contextualize or integrate a table in our database with data from an external source.

We want to learn how to take data from different tables and combine it together. This may include data from other data sources that complement our own, such as demographic information for a zip code or price structure for shipping zones for a carrier. The process of merging two separate tables is called "joining". Joins may be done on any columns in two tables, as long as the merge operation makes logical sense. See below for a visual representation of joining:



Columns in a JOIN

- They don't need to be keys, though they usually are
- Join columns must have compatible data types
- Join column is usually key column: Either primary or foreign
- NULLs will never join

Types of Joins and Views

Join from 3 Tables: Joining three tables together just involves one additional join between two already joined tables and a third table.

This work is licensed under a Creative Commons Attribution-NonCommercial-ShareAlike 4.0 International License.

Join Types

Different types of joins can be used to merge two tables together that always include every row in the left table, right table, or in both tables. The following are different types of JOIN:

- INNER JOIN: returns only the records with matching keys (joins common column values)
- LEFT JOIN: returns all rows from LEFT (first) table, whether or not they match a record in the second table
- RIGHT JOIN: returns all rows from RIGHT (second) table, whether or not they match a record in the first table
- OUTER JOIN: Returns all rows from both tables, whether or not they match (Microsoft SQL, not MySQL)
- In MySQL, JOIN and INNER JOIN are equivalent

Views

Views are virtual tables that do not change the underlying data but can be helpful to generate reports and simplify complicated queries. They are virtual tables that present data in a denormalized form to users. They do no create separate copies of the data (they reference the data in the underlying tables). The database stores a definition of a view and the data is updated each time the VIEW is invoked.

There are several advantages to VIEWS. User queries are simpler on views constructed for them. They offers a layer of security that can restrict access to data in views for users. They also provide greater independence, meaning that the user or program is not affected by small changes in underlying tables.

Learning Objectives

- Learn how to work with SELECT for conditional clauses.
- Recognize the role and use of NULL values.
- Review how to group data with the GROUP BY clause.
- Introduce the existence of statistical functions in all SQL functions.
- Recognize how to apply aggregate statistical functions.
- Review sorting and sampling techniques such as ORDER BY, LIMIT and RAND.
- Learn how to create new tables and aliases using the AS Keyword.
- Becoming familiar with joining multiple tables.
- Recognize the types of JOINs.
- Identify when and how to use VIEWs.

This work is licensed under a Creative Commons Attribution-NonCommercial-ShareAlike 4.0 International License.