SQL Primer

Contents

[1 Introduction 1](#_Toc195098534)

[2 The *SELECT* Statement 2](#_Toc195098535)

[3 The *INSERT* Statement 4](#_Toc195098536)

[4 The *UPDATE* Statement 4](#_Toc195098537)

[5 The *DELETE* Statement 5](#_Toc195098538)

[6 Linked Tables 6](#_Toc195098539)

[7 The INNER JOIN Clause 7](#_Toc195098540)

[Appendix 1 Inner Join – Multiple Linked Tables 9](#_Toc195098541)

# Introduction

[SQL](http://en.wikipedia.org/wiki/SQL) (Structured Query Language) is the language we use to communicate instructions to a database. This document provides a brief overview of SQL. In a later document we consider how to use PHP to issue SQL statements. A *SQL statement* is a string which describes what we want to do to a database. There are four types of SQL statements[[1]](#footnote-1):

1. [DML](http://en.wikipedia.org/wiki/Data_manipulation_language) (Data Manipulation Language) – Used to retrieve, store, modify, and delete data in database. Examples: SELECT, INSERT, UPDATE, DELETE statements
2. [DDL](http://en.wikipedia.org/wiki/Data_definition_language) (Data Definition Language) - Used to create and modify the structure of database objects in database. Examples: CREATE, ALTER, DROP statements
3. [DCL](http://en.wikipedia.org/wiki/Data_control_language) (Data Control Language) – Used to create roles, permissions, and referential integrity as well as to control access to database by securing it. Examples: GRANT, REVOKE statements
4. [TCL](https://byjus.com/gate/transaction-control-language-tcl-notes/) (Transactional Control Language) – Used to manage different transactions occurring within a database. Examples: COMMIT, ROLLBACK statements

In this course we will only consider DML and specifically the four statements: *Select, Insert, Update,* and *Delete*. A *Select* SQL statement *returns records* from a database. *Insert, Update,* and *Delete* statements do not return any data, they perform an action on the database by inserting new record, updating an existing record, or deleting a record, respectively. These three types of SQL statements are called a*ction queries*. These four SQL statements are the basic for *CRUD* applications (Create, Retrieve, Update, Delete).

A good [SQL Tutorial](https://www.w3schools.com/sql/default.asp) is found at w3schools:

# The *SELECT* Statement

Good tutorial and/or reference on MySQL: <https://www.w3resource.com/mysql/mysql-tutorials.php>

The purpose of a *SELECT* statement is to return rows of a database table. You can return all rows, or you can filter rows. You can also control which fields (columns) you return. In the examples below, I will use all CAPS for SQL keywords (however, keywords are not case-sensitive) and I will use *italics* for names of fields and tables. A SQL statement is a string. Sometimes you can hard-code the entire string. Other times you must build it programmatically.

1. The most basic syntax for the SELECT statement:

SELECT \* FROM *TableName*

Which selects all the data (rows and columns) from the table, *TableName*. The data can be returned, depending on options used, in various formats: an array of rows, where the indices into the row can be indices or keys; an array of objects of some class, and others.

1. Restricting Columns – Often, we don't need all the columns/fields in a table. Thus, we can specify just the fields that we want:

SELECT *fld\_1*, *fld\_2*, ... FROM *TableName*

Example – Select the player ID, last name, first name and birth date from the Players table

SELECT PlayerID, LName, FName, BDate FROM Players

1. Ordering the Results – We can use the *ORDER BY* clause to order (sort) the data that is returned. Without an *Order By* clause, the data is returned in the order that it was inserted to the table.

SELECT *fld\_1*, *fld\_2*, ...

FROM *TableName*

ORDER BY *fld\_i* ASC, *fld\_k* DESC, ...

Returns the data sorted by *fld\_i* ascending, followed by *fld\_k* descending. If "ASC" or "DESC" is not specified, then the default is "ASC". Examples:

SELECT PlayerID, LName, FName, BDate FROM Players ORDER BY LName, FName

SELECT PlayerID, LName, FName, BDate FROM Players ORDER BY BDate DESC

1. Restricting Rows – We can use the *WHERE* clause to be selective about which rows are returned. The general syntax is:

SELECT *list\_of\_fields* FROM *table*

WHERE *condition1*, *condition2*, ...

ORDER BY ...

Examples:

1. Select the employees who make more than $50/hr.

SELECT *EmpID, Name, PayRate* FROM *Employees*

WHERE *PayRate > 50.0*

ORDER BY *PayRate* DESC, *Name* ASC

1. Other Examples:

SELECT \* FROM Employees WHERE EmpID = 'YQ4452' // String field

SELECT \* FROM Employees WHERE HireDate > #5/1/99# // DateTime field

SELECT \* FROM Employees WHERE PayRate >= 20.00 // Numeric field

Note that strings must be delimited by single quotes, dates with “#” for MS Access and SQL Server. I’m not sure how MySQL handles them. Numbers (integer or decimal) are not delimited.

1. Compound Conditions – Composed with a logical operator (AND, OR, NOT, and others) and the use of parentheses (when needed). Example:

SELECT \* FROM *Employees*

WHERE *PayRate* > 50.0 AND *HireDate* <= #3/1/01#

1. BETWEEN Operator – The *BETWEEN* operator is inclusive. Examples:
2. Select employees whose pay rate is between $10 and $30 per hour and their hire date is between March 1, 1999 and March 1, 2000

SELECT \* FROM *Employees*

WHERE (*PayRate* BETWEEN 10.0 AND 30.0) AND

(*HireDate* BETWEEN #3/1/99# AND #3/1/00#)

1. Select employees whose pay rate is less than $10 and more than $30 per hour

SELECT \* FROM *Employees*

WHERE *PayRate* NOT BETWEEN 10.0 AND 30.0

1. LIKE Operator – The *LIKE* operator is used for pattern matching. For instance, you may want to return only rows whose Name field begins with "G". To do this we also need *string* operators:

“%” – Matches zero or more characters

“\_”– Matches exactly one character

Note that mySql also supports *RLIKE* which is followed by a regular expression.

Examples:

1. Select all last names that begin with "G", including the last name "G" itself, if it exists.

SELECT \* FROM *Employees* WHERE *LName* LIKE 'G%'

1. Select all 4-character names that begin with “G”.

SELECT \* FROM *Employees* WHERE *LName* LIKE 'G\_\_\_'

1. Select all names that are at least 2 characters in length whose second character is “u”.

SELECT \* FROM *Employees* WHERE *LName* LIKE '\_u%'

1. You can also use the LIKE operator to select a range of characters.

Examples:

1. Select all last names that begin with the letters “A” through "G".

SELECT \* FROM Employees WHERE *LName* LIKE '[A-G]%'

1. Select all last names whose second letter is a vowel or who make less than or equal to $50/hr.

SELECT \* FROM Employees

WHERE (*LName* LIKE '\_[a,e,i,o,u]%') OR (*PayRate* <= 50.0)

1. IN Operator – The *IN* operator allows you to specify multiple values in a WHERE clause. Example: Select all players with a jersey number of 33, 45, or 49.

SELECT \* FROM Players

WHERE *JerseyNum* IN (33,45,49) ORDER BY *JerseyNum*

1. Summary of operators:
2. Basic comparison operators: <, <=, =, >=, >, <>.
3. Basic logical operators: AND, OR, NOT, BETWEEN, LIKE, IN.
4. Basic string operators: %, \_,

Also, note some databases use “\*” instead of “%” and “?” instead of “\_”.

A good reference: [Use wildcards in queries and parameters in Access](https://support.microsoft.com/en-us/office/use-wildcards-in-queries-and-parameters-in-access-ec057a45-78b1-4d16-8c20-242cde582e0b). Note: the wildcards shown are different than the ones in these notes; however, the ones I use do work.

# The *INSERT* Statement

1. The *INSERT* statement inserts a new row into a table. The syntax is:

INSERT INTO *table* ( *fld1*, *fld2*, ... ) VALUES ( *val1*, *val2*, ... )

1. Consider an *Employees* table with fields: *EmpID, Name, Date, Pay.* To insert a row:

INSERT INTO Employees (Name, Date, Pay) VALUES ( 'DG', #1/1/01#, 9.23 )

1. **Note we never specify the value of the primary key with an Insert statement. This is taken care of by the database engine.**

# The *UPDATE* Statement

1. The *UPDATE* statement is used to modify the data in a row. The syntax is:

UPDATE *table* SET *fld1*=*val1*, *fld2*=*val2*, ... WHERE *criteria*

1. Examples
2. Set the salary of the employee with ID=483 to $100:

UPDATE Employees SET Pay=100.0 WHERE EmpID=483

1. Give the employee with ID=483 a 10% raise:

UPDATE Employees SET Pay=Pay\*1.1 WHERE EmpID=483

1. Give everyone in Department 15, a 10% raise:

UPDATE Employees SET Pay=Pay\*1.1 WHERE DeptID=15

1. **Note we never update the value of the primary key with an Update statement; it cannot be changed. However, frequently we will use it in the WHERE clause to select the row we want to update.**

# The *DELETE* Statement

1. The *DELETE* statement is used to remove rows from a table. The syntax for the DELETE statement is:

DELETE FROM *table* WHERE *criteria*

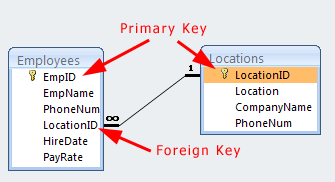
1. Examples:
2. Delete the employee with ID=483

DELETE FROM Employees WHERE EmpID = 483

1. Delete all employees who were hired after 1/24/2025 (DOGE firing date).

DELETE FROM Employees WHERE hireDate > #1/24/2025#

# Linked Tables

1. The diagram on the right shows the relationships between an *Employees* table and a *Locations* table in a database.
2. The black line that connects the two tables establishes a relationship between the two tables. It shows that each *Employee* has exactly one *Location* and each *Location* can have many *Employees*.
3. The relationship is established through a *primary key – foreign key relationship*. The *LocationID* is a *primary key* in the *Locations* table *a foreign key* in the *Employees* table.
4. The *Locations* table is sometimes referred to as a *lookup* table. We use it to *lookup* information about an *Employees* location.
5. The relationshipbetween these two tables is said to enforce *referential integrity*. This means that when you add an Employee you must assign them to a valid (existing) Location (for SQL Server and Access, some databases allow you to assign NULL to the foreign key value). For example, if the *Locations* table contained this data:

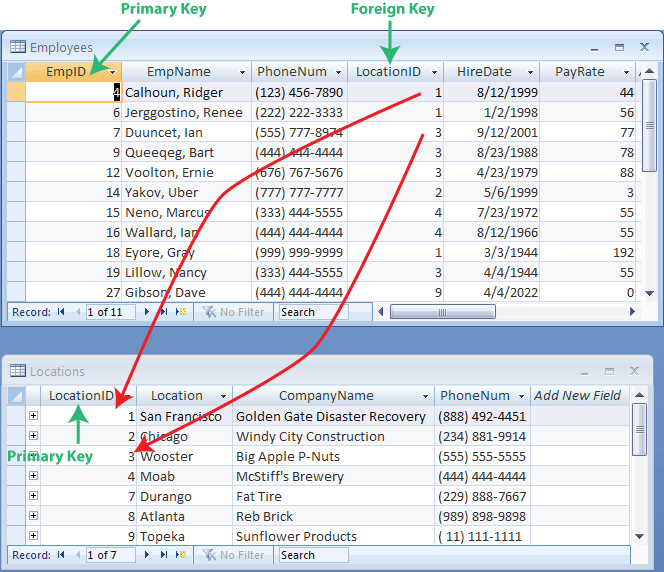
|  |  |  |  |
| --- | --- | --- | --- |
| **LocationID** | **Location** | **CompanyName** | **PhoneNum** |
| 1 | Atlanta | Synergistic | 232-8821 |
| 2 | Macon | Lotine | 332-2393 |
| 3 | Athens | REMDogs | 558-4728 |

And we tried to add a new player to Team 5 (LocationID=5):

**Insert Into Employees (EmpName, LocationID) Values (‘Dave’, 5)**

It would fail because there is not a LocationID=5. Similarly, it would fail if we used this statement:

**Insert Into Employees (EmpName, PayRate) Values (‘Dave’, 55.55)**

because we did not supply a LocationID.

1. Some databases do not enforce referential integrity; it is left up to the programmer. SQL Server and Access do enforce referential integrity.
2. The data in these tables is shown on the right with indications of how the rows in the *Employees* table are linked to rows in the lookup (*Locations*) table.

# The INNER JOIN Clause

1. In the example above, suppose that we want a list of all employees, their locations, and the phone number of the main office at that location. This information is contained in two tables. An *Inner Join* clause is the preferred technique to combine information from two or more tables that have a relationship between a primary key and a foreign key. The way it works is that you *join* the two tables *on* the foreign key - primary key. This, effectively, combines the two tables into one. There are other types of joins[[2]](#footnote-2) [[3]](#footnote-3) (left, right, full) but we will not consider these.
2. Example 1 – To obtain all employees and their corresponding locations, we might write a SQL statement like this:

**SELECT Employees.EmpName, Employees.PhoneNum, Employees.HireDate,**

**Employees.PayRate, Locations.Location, Locations.CompanyName,**

**Locations.PhoneNum**

**FROM Employees**

**INNER JOIN Locations**

**ON Employees.LocationID = Locations.LocationID;**

Note:

* When selecting form multiple tables, we must use the syntax: *TableName.FieldName* to select fields to differentiate two tables with the same field name. For example, above, The *Employees* table has a *PhoneNum* that represents the phone number of the employee. And, the *Locations* table also has a *PhoneNum* filed that represents the phone number of the main office for that location.
* We did not select the primary key (*EmpID*) and foreign key (*LocationID*) from the Employees table. Frequently we will want to do this. For instance, if we later wanted to update an employee’s data we would need the *EmpID*.

1. Example 2 – Of course, we can also apply a WHERE clause to the INNER JOIN. For instance, we may only want to select the employees from the Chicago location (LocationID=2). We would do this by attaching this clause to the end of the statement above:

**WHERE Employees.LocationID=2**

1. Example 3 – Another way to return the same data as Example 1 is:

**SELECT Employees.EmpName, Employees.PhoneNum, Employees.HireDate,**

**Employees.PayRate, Locations.Location, Locations.CompanyName,**

**Locations.PhoneNum**

**FROM Employees, Locations**

**WHERE Employees.LocationID = Locations.LocationID;**

Note:

* No inner join was used. Instead, the FROM clause specifies the two tables.
* **However, the preferred[[4]](#footnote-4) [[5]](#footnote-5) [[6]](#footnote-6)method is to use an INNER JOIN. You will be responsible for the preferred approach on tests and assignments.**

1. Example 4 – Once we introduce the INNER JOIN, I find that students think that we **always** need an INNER JOIN when we have linked tables. **This is not so**. It completely depends on what your use case is. For instance, suppose that I want just the names of the employees at the Chicago location. Solution: I need to select from the *Employees* table but use the WHERE clause to specify only employees from Chicago (*LocationID=2*):

**SELECT Employees.EmpID, Employees.EmpName, Employees.PhoneNum,**

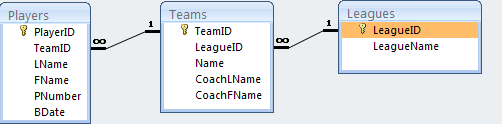
**Employees.LocationID**

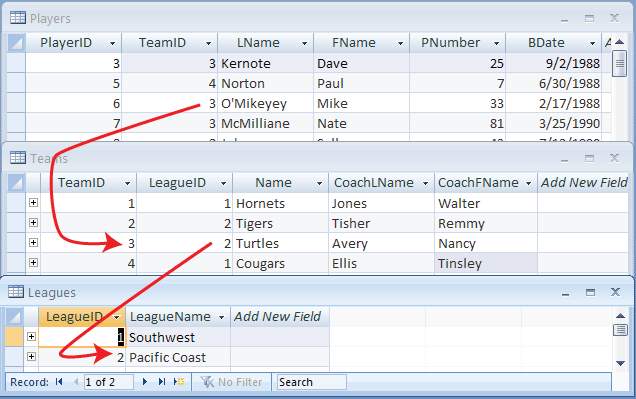
**FROM Employees**

**WHERE Employees.LocationID=2**

Appendix

1. Inner Join – Multiple Linked Tables
2. Another example with several linked tables: Each player has one team and each team has on league. Or, each league has many teams and each team has many players.





1. Example 5 – To select all the teams and the corresponding leagues:

**SELECT Teams.TeamID, Teams.LeagueID, Teams.Name, Teams.CoachLName,**

**Teams.CoachFName, Leagues.LeagueName**

**FROM Teams**

**INNER JOIN Leagues**

**ON Teams.LeagueID = Leagues.LeagueID**

1. Example 6 – To select all the players, their corresponding team and league:

**SELECT Players.LName, Players.FName, Teams.Name, Leagues.LeagueName**

**FROM (Players INNER JOIN Teams ON Players.LeagueID = Teams.LeagueID)**

**INNER JOIN Leagues**

**ON Teams.TeamID = Leagues.TeamID**

1. Example 7 – To select all the players with birth dates on or after 5/1/1999 from the Pacific Coast League:

**SELECT Players.LName, Players.FName, Players.BDate, Teams.Name,**

**Leagues.LeagueName, Leagues.LeagueID**

**FROM (Players INNER JOIN Teams ON Players.LeagueID = Teams.LeagueID)**

**INNER JOIN Leagues**

**ON Teams.TeamID = Leagues.TeamID**

**WHERE (((Players.BDate)>=#5/1/1999#) AND ((Leagues.LeagueID)=2))**

1. Source: (almost directly copied from): <http://blog.sqlauthority.com/2008/01/15/sql-server-what-is-dml-ddl-dcl-and-tcl-introduction-and-examples/> [↑](#footnote-ref-1)
2. <http://www.w3schools.com/sql/sql_join.asp> [↑](#footnote-ref-2)
3. <http://en.wikipedia.org/wiki/Join_%28SQL%29> [↑](#footnote-ref-3)
4. <http://stackoverflow.com/questions/2241991/in-mysql-queries-why-use-join-instead-of-where> [↑](#footnote-ref-4)
5. <http://stackoverflow.com/questions/44917/explicit-vs-implicit-sql-joins> [↑](#footnote-ref-5)
6. <http://stackoverflow.com/questions/121631/inner-join-vs-where> [↑](#footnote-ref-6)