



# SQL

---

Data Science Bootcamp

---

# Overview

---

- ❖ **Intro to SQL**
  - ❖ Tables and schemas
  - ❖ SQL queries - SELECT
  - ❖ MySQL database management
    - Databases
    - Creating and populating tables
- ❖ Joins

## SQL and relational databases

---

- ❖ A *relational database management system* (RDBMS) is a database consisting of *relations*, or *tables*, using SQL as its *query language*.
- ❖ SQL - Structured Query Language - is a *standardized* language for expressing *queries* - questions for the database. Nearly all RDBMS's use a standard form of SQL.
- ❖ There are many RDBMSs; much of the business world runs on RDBMS's from Oracle, IBM, MS, and other large companies.
- ❖ Conversely, most of the internet uses open-source databases. We will work with **MySQL**, a very popular open-source RDBMS.
- ❖ Find the full documentation at [dev.mysql.com/doc/refman/5.7/en/sql-syntax.html](https://dev.mysql.com/doc/refman/5.7/en/sql-syntax.html)

---

# Overview

---

- ❖ Intro to SQL
- ❖ **Tables and schemas**
- ❖ SQL queries - SELECT
- ❖ MySQL database management
  - Databases
  - Creating and populating tables
- ❖ Joins

# Tables

---

- ❖ Data is organized into named *tables*, or *relations*.
- ❖ A table is a rectangular grid with *column names*. Each location in the grid is called a *cell*. Cells contain *values*, such as character strings, numbers, or dates.

Admissions		
<i>Facility</i>	<i>IllnessCode</i>	<i>Lengthofstay</i>
St. Marks	141	2
Mercy	152	4
Mercy	94	7
Jay County	152	5

Illnesses	
<i>Code</i>	<i>Description</i>
94	Tonsillitis
141	Asthma
152	Broken arm
160	Appendicitis

# Schemas

- ❖ The names and types of the data in a table's columns are called the table's *schema*.
- ❖ These are the schemas of the tables on the previous slide:

Admissions		
<i>Facility</i>	<i>IllnessCode</i>	<i>Lengthofstay</i>
char(30)	int	int

Illnesses	
<i>Code</i>	<i>Description</i>
int	char(30)

- ❖ You can use any names for the columns. There are over 30 data types, but these are the most widely used:
  - int - also different-sized ints, like tinyint, smallint, bigint
  - float
  - decimal(*p*, *s*) - number with *p* digits, of which *s* are after the dec pt.
  - char(*size*) - character string of length *size*
  - boolean - value that is just "TRUE" or "FALSE"

## Exercise 1 - Log in and start MySQL

- ❖ Login to MySQL server node (18.220.240.108).
  - Mac OS: `ssh yourname@18.220.240.108`
  - Windows: Download [Putty](#) to login (direct download link).
- ❖ Look at the tables. Go into MySQL and run the following commands:

```
$ mysql
mysql> SHOW DATABASES;
mysql> USE sqlexercise;
mysql> SHOW TABLES;
mysql> DESCRIBE hospital;
mysql> DESCRIBE DiagCodeMap;
```

---

# Overview

---

- ❖ Intro to SQL
- ❖ Tables and schemas
- ❖ **SQL queries - SELECT**
- ❖ MySQL database management
  - Databases
  - Creating and populating tables
- ❖ Joins



# SQL

---

- ❖ SQL is a standardized language for writing queries, e.g. “give me the description of illness code 152,” or “give me the code for asthma,” or “give me the average length of stay for a patient with a broken arm,” or “tell me the illness descriptions for all patients admitted to Mercy.”
- ❖ SQL queries are typed into a database command interpreter, or *query engine*, and the results either printed or stored in a file.
- ❖ SQL is quite a complicated language, but the most common features are not difficult to use. We will discuss:
  - **Projection** - selecting some columns from a table
  - **Function application** - applying a function to a value
  - **Filtering** - selecting rows based on some criterion
  - **Grouping** - aggregating rows based on the values in a column
  - **Joins** - combining two tables

## Query syntax

---

- ❖ SQL queries have the form:

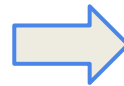
```
SELECT expression [, expression, ...]  
FROM table  
[... additional clauses ...] ;
```

- ❖ The square brackets mean “optional.” You have to have at least one expression after the SELECT, but you can have as many as you want. The table must be named in a FROM clause, but the simplest queries have no additional clauses.
- ❖ The result of a query can itself be considered as a table. The number of columns will be equal to the number of expressions after “SELECT”.
- ❖ The division into lines is not significant. It is best to break up lines if they are too long, just to make queries more readable.

# Projection

- ❖ Projection is when you ask for only some of the columns of a table.

```
SELECT Facility  
FROM Admissions;
```



<i>Facility</i>
St. Marks
Mercy
Mercy
Jay County

- ❖ Projections often produce identical rows. If you don't want them, use SELECT DISTINCT:

```
SELECT DISTINCT Facility  
FROM Admissions;
```

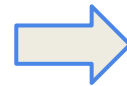


<i>Facility</i>
St. Marks
Mercy
Jay County

# Projection

- ❖ You can include as many columns as you like:

```
SELECT Facility, IllnessCode  
FROM Admissions;
```



<i>Facility</i>	<i>IllnessCode</i>
St. Marks	141
Mercy	152
Mercy	94
Jay County	152

- ❖ Use “\*” as an abbreviation for listing all columns:

```
SELECT *  
FROM Admissions;
```



<i>Facility</i>	<i>IllnessCode</i>	<i>Lengthofstay</i>
St. Marks	141	2
Mercy	152	4
Mercy	94	7
Jay County	152	5

## Syntactic notes

---

❖ It is conventional - but not required - to use capital letters for *keywords* like SELECT and FROM. In MySQL, case matters in table names, but not in keywords or column names.

➤ SELECT Facility FROM Admissions - *good*

➤ SELECT Facility FROM admissions - *bad*

➤ select facility FROM Admissions - *good*

➤ SeLeCt Facility FROM Admissions - *good*

*(In some installations, case does not matter in table names either; this is a configurable feature.)*

## Exercise 2 - Projection

- ❖ Execute the following queries. You should be at the “mysql>” prompt, in database `sqlexercise`.
  - `SELECT DiagnosisCode, Severity FROM hospital;`
  - `SELECT * FROM DiagCodeMap;`
  - List all payment sources from the hospital table, without repetitions.
  - List the diagnosis code and severity of each record, first with repetitions, then without.

## Function application

- ❖ Instead of just reporting the value in a cell, apply a function to it:

```
SELECT Facility,  
       Lengthofstay * 24  
FROM Admissions;
```



<i>Facility</i>	
St. Marks	48
Mercy	96
Mercy	168
Jay County	120

```
SELECT UPPER(Facility),  
       Lengthofstay * IllnessCode  
FROM Admissions;
```



ST. MARKS	282
MERCY	608
MERCY	658
JAY COUNTY	760

## Column names in SELECT statements

- ❖ In the examples on the previous slide, the query results sometimes have column names and sometimes don't.
- ❖ The rule is: If you select a column, SQL will use its name as the column name in the result. But if the new column is the result of a calculation, SQL will make up a name for the column *that you cannot rely upon*.
- ❖ To give a name to a calculated column, use AS:

```
SELECT Facility,  
       Lengthofstay * 24 AS stayinhours  
FROM Admissions;
```



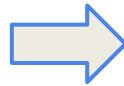
<i>Facility</i>	<i>stayinhours</i>
St. Marks	48
Mercy	96
Mercy	168
Jay County	120



# Filtering

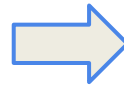
- ❖ Select rows based on a condition

```
SELECT *  
FROM Admissions  
WHERE Lengthofstay > 4;
```



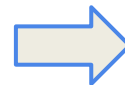
<i>Facility</i>	<i>IllnessCode</i>	<i>Lengthofstay</i>
Mercy	94	7
Jay County	152	5

```
SELECT Facility, IllnessCode  
FROM Admissions  
WHERE IllnessCode = 141 OR  
       IllnessCode = 94;
```



<i>Facility</i>	<i>IllnessCode</i>
St. Marks	141
Mercy	94

```
SELECT Facility, Lengthofstay  
FROM Admissions  
WHERE Lengthofstay > 2 AND  
       Lengthofstay < 6;
```



<i>Facility</i>	<i>Lengthofstay</i>
Mercy	4
Jay County	5

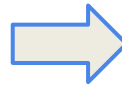
## Exercise 3 - Filtering

- ❖ You should be at the “mysql>” prompt, in database sqlexercise. Answer the following questions by using the hospital table.
  - a. List all rows where admissiontype is “Urgent”.
  - b. Same as (a), but add a column chargeperday (chargeperday = totalcharges / lengthofstay).
  - c. List the ages (actually, age ranges) of all patients with illness of severity > 2, without repetitions.

## Grouping

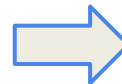
- ❖ Aggregate the rows according to a column, and perform operations on the aggregated rows.

```
SELECT Facility,  
       MAX(Lengthofstay)  
FROM Admissions  
GROUP BY Facility;
```



<i>Facility</i>	
St. Marks	2
Mercy	7
Jay County	5

```
SELECT Facility,  
       MIN(Lengthofstay),  
       MAX(Lengthofstay),  
       AVG(Lengthofstay)  
FROM Admissions  
GROUP BY Facility;
```



<i>Facility</i>			
St. Marks	2	2	2
Mercy	4	7	5.5
Jay County	5	5	5

## Aggregate functions

---

- ❖ SQL has numerous functions that can be applied to a column from a group when using the GROUP BY clause.
- ❖ Implementations vary, but some of the the most popular functions are:
  - MIN, MAX - minimum or maximum of values in the column for that group; applies to numbers or strings.
  - SUM, AVG, STDEV - sum, average, and standard deviation; apply only to numbers
  - COUNT - size of the group; you will get the same answer if you write COUNT(*column*), for any *column*, or just COUNT(\*).

## Aggregate functions

---

- ❖ The aggregate functions can also be used to find overall values, i.e. without a GROUP BY clause.
  - Find the number of rows in a table:

```
SELECT COUNT(*) FROM Admissions;
```

## Grouping and filtering

- ❖ GROUP BY and WHERE can be used together:

```
SELECT Facility,  
       MIN(Lengthofstay)  
FROM Admissions  
WHERE Lengthofstay > 4  
GROUP BY Facility;
```



<i>Facility</i>	
Mercy	7
Jay County	5

Filtering happens before grouping. This query gives us the minimum length of all stays greater than 4 days, for each facility. (Syntactically, the WHERE clause must precede the GROUP BY clause.)

## Filtering on aggregate values

- ❖ You can filter by an aggregate value using the HAVING clause:

```
SELECT Facility,  
       MIN(Lengthofstay)  
FROM Admissions  
GROUP BY Facility  
HAVING MIN(Lengthofstay) > 4;
```



<i>Facility</i>	
Jay County	5

- ❖ Mercy is not included in this result because its minimum stay is not greater than 4 (it's 4).

## Exercise 4 - Grouping

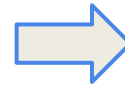
- ❖ You should be at the “mysql>” prompt, in database sqlexercise. Answer the following questions by using the hospital table.
  - How many males and females are there? (The result will be a two-row table.)
  - Find the average length of stay of all admissions, aggregated by severity. (Use function `AVG(lengthofstay)` in the `SELECT`.)
  - Find the average total cost of admissions, aggregated by both gender and severity. (To group by two columns, just give both columns in the `GROUP BY` clause, separated by a comma.)



## Getting sorted output

- ❖ Sort the result of a query using the ORDER BY clause.

```
SELECT Facility, Lengthofstay  
FROM Admissions  
ORDER BY Lengthofstay;
```



<i>Facility</i>	<i>Lengthofstay</i>
St. Marks	2
Mercy	4
Jay County	5
Mercy	7

```
SELECT Facility, Lengthofstay  
FROM Admissions  
ORDER BY Facility, Lengthofstay;
```

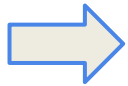


<i>Facility</i>	<i>Lengthofstay</i>
Jay County	5
Mercy	4
Mercy	7
St. Marks	2

# Joins

- ❖ Joins are the one operation that combines multiple tables.
- ❖ **Idea:** Give a condition involving columns from two tables. Whenever the condition is true for any pair of rows (one from the first table and one from the second), combine those two rows into one long row and include it in the output.

```
SELECT * FROM Admissions JOIN Illnesses  
ON Admissions.IllnessCode = Illnesses.Code;
```



<i>Facility</i>	<i>IllnessCode</i>	<i>Lengthofstay</i>	<i>Code</i>	<i>Description</i>
St. Marks	141	2	141	Asthma
Mercy	152	4	152	Broken arm
Jay County	152	5	152	Broken arm

# Joins

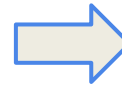
---

- ❖ Joins are by far the most expensive kind of query to execute, because they involve matching every row in one table with every row in another. *Query optimization* is a process that query engines do to execute queries faster.
- ❖ You can help optimize your queries by avoiding joining large tables. For example, if you can, filter *before* you join instead of after.
- ❖ There are several variations on joins. The one we just discussed is called “inner join,” or just “join.” We’ll later discuss “outer joins” (and their variants).

## LIMIT clause

- ❖ A really handy feature of SQL is the LIMIT clause, which restricts the number of rows in the result of a query.

```
SELECT Facility, IllnessCode  
FROM Admissions  
LIMIT 2;
```



<i>Facility</i>	<i>IllnessCode</i>
St. Marks	141
Mercy	152

- ❖ The LIMIT clause is useful when you just want to get a general idea of what the result will be, and when you are debugging a query.
- ❖ *Important note:* Just because you are producing a smaller result does not necessarily mean the query engine is doing less work. For example, if you add a LIMIT clause to a JOIN, the query engine may still have to do the entire JOIN (which is expensive).

---

# Overview

---

- ❖ Intro to SQL
- ❖ Tables and schemas
- ❖ SQL queries - SELECT
- ❖ MySQL database management

## ➤ Databases

- Creating and populating tables
- ❖ Joins

# Managing databases

---

- ❖ We have seen how to query existing tables (and we will do some more of that today). This leaves the question of how to *create* and *update* tables.
- ❖ Today we will talk about:
  - Databases - groups of related tables
  - Creating and populating tables
  - Updating tables
  - Deleting tables
- ❖ Then we'll talk some more about joins.

# Databases

---

- ❖ Databases are simply groups of related tables.
  - You have been working in the `sqlexample` database which we created. We also have databases for each of you, which you will be working in today.
- ❖ You can only create a database if you are an administrator. For the record, the commands to create and delete a database are: `CREATE DATABASE database-name` and `DROP DATABASE database-name`.
- ❖ To list the existing databases, enter: `SHOW DATABASES;`
- ❖ To use a database, enter: `USE database-name;`
- ❖ To see the tables in a database you are using, enter: `SHOW TABLES;`
- ❖ To see the schema of a table, enter: `DESCRIBE table-name;`

## Exercise 5 - Listing tables

- ❖ Quit MySQL by typing “quit;”. Then start again by typing MySQL.
- ❖ Enter these commands:
  - `SHOW DATABASES;`
  - `USE sqlexercise;`
  - `SHOW TABLES;`
  - `USE yourname_db;`
  - `SHOW TABLES;`



---

# Overview

---

- ❖ Intro to SQL
- ❖ Tables and schemas
- ❖ SQL queries - SELECT
- ❖ MySQL database management
  - Databases
  - **Creating and populating tables**
- ❖ Joins

## Creating tables

---

- ❖ If you're using a database in which you have the right permissions, you can create tables.
- ❖ Tables are built in two steps: First *create* the table: give it a name and schema; then *populate* it, either by importing a file or by entering data directly at the command line.
- ❖ The command to create a table is:

```
CREATE TABLE table-name ( schema );
```

where *schema* is a comma-separated list of column names and types.

- ❖ For example:

```
CREATE TABLE diagcodes (code int(3), descr char(30));
```

## Deleting rows and tables

---

- ❖ To remove a table entirely, use

```
DROP TABLE table-name ;
```

- ❖ To delete the *data* of a table but leave the table (as a schema), use

```
TRUNCATE TABLE table-name ;
```

- ❖ To delete rows of a table that satisfy a WHERE clause use

```
DELETE FROM table-name  
WHERE condition ;
```

## A note on data types

---

- ❖ When giving the schema of a new table, every type includes the length designator (number in parentheses). Its meaning depends upon the type. (If omitted, a default value will be given.)
  - `char(size)`: *size* is the exact size of the field; if the number of real characters is less than *size*, it is filled with spaces. (*size* defaults to 1.)
  - `int(size)`: *size* represents the number of columns used to display the field. The actual internal “word size” is usually 32 or 64 bits, i.e. the usual size of a computer word. (*size* defaults to 11.)
  - `decimal(p, s)`: *p* is the exact size in digits; *s* is the number of those digits considered as being after the decimal point. *p* and *s* default to 10 and 0.

## Populating tables

---

- ❖ Now that you've created a table, you can add rows in one of two ways:
  - `INSERT INTO` is used to add records in an SQL command.
  - `LOAD DATA` is used to add data from a file.
- ❖ `UPDATE` is used to change a value in an existing row.

## Creating a table

---

- ❖ In the Linux class, you copied a file into your home directory called iris.csv. It should still be there. Here are the first few lines of that file:

```
"Sepal.Length", "Sepal.Width", "Petal.Length", "Petal.Width", "Species"  
5.1,3.5,1.4,0.2,"setosa"  
4.9,3,1.4,0.2,"setosa"
```

- ❖ An appropriate command to create a table for this type of data is:

```
CREATE TABLE iris (  
    SepalLength float,  
    SepalWidth float,  
    PetalLength float,  
    PetalWidth float,  
    Species char(255));
```

## Exercise 6 - Creating a table

- ❖ Make sure you are in *your* database. (In MySQL, enter: `show tables;` The result should be a table with name `Tables_in_yourname_db.`)
- ❖ Enter the CREATE TABLE statement on the previous slide.
- ❖ You should get a “Query OK” response from MySQL.
- ❖ Enter: `DESCRIBE iris;`
- ❖ If you enter the CREATE TABLE statement incorrectly, here are two tips:
  - Use the up-arrow key to recall the query. You can then edit it: use the left- and right-arrow keys to move within it, and backspace to delete characters. Then hit enter again.
  - If you created the table but it’s wrong, enter: `DROP TABLE iris;` and start over.

## Populating a table from a file

---

- ❖ We'll populate the iris table directly from iris.csv using the LOAD DATA command. This command has numerous options. Here's the version we'll use:

```
LOAD DATA LOCAL INFILE 'iris.csv'  
INTO TABLE iris  
FIELDS TERMINATED BY ','  
OPTIONALLY ENCLOSED BY '"'  
LINES TERMINATED BY '\n'  
IGNORE 1 lines;
```

- ❖ This is mostly self-explanatory.
  - The text portion of each row is enclosed in double quotes. It is not actually needed here, but is used when data may contain commas.
  - The last line is to account for the header line in the csv file.



## Populating a table from a command

---

- ❖ You can add records to a table with the INSERT INTO command.

```
INSERT INTO iris  
VALUES (5.8, 3.0, 5.7, 2.0, 'carolinica'),  
       (5.3, 3.2, 5.6, 2.3, 'marylandica');
```

## Updating a record

---

- ❖ Update fields of all records that match a WHERE clause:

```
UPDATE iris  
SET PetalWidth = 0.3,  
    PetalLength = 1.7  
WHERE Species = 'marylandica';
```

- ❖ This will update all records that match the WHERE clause (so be careful).

## Exercise 7 - Populating a table

- ❖ Populate the iris table you created, using the commands given on the previous slides.
- ❖ Initialize the iris table using the LOAD DATA command.
  - If you don't have `iris.csv`, copy it again:  

```
cp /mnt/data/iris.csv .
```
- ❖ Add a new row using INSERT INTO.

## Creating a table from a query

---

- ❖ There is another way to create a new table: give a name to the result of a query. The syntax is:

```
CREATE TABLE table-name  
AS select-statement ;
```

The schema for the new table is inferred from the select statement.

- ❖ Example:

```
CREATE TABLE longstays  
AS SELECT facility, MAX(lengthofstay)  
FROM hospital  
GROUP BY facility;
```

## Subqueries

---

- ❖ Instead of creating a new table from a SELECT and then referring to that table in another query, you can just use the SELECT within the main query in place of the table reference. That is, instead of

```
CREATE TABLE avg_costs AS SELECT select-statement1
```

```
SELECT select-statement2 FROM avg_costs
```

write

```
SELECT select-statement2 FROM SELECT select-statement1
```

## Subqueries

---

- ❖ You will often find that you need to distinguish the columns in the new table from the columns in the old one. To give the subquery result a name, use this syntax:

```
SELECT select-statement2  
FROM (SELECT select-statement1) name
```

or

```
SELECT select-statement2  
FROM (SELECT select-statement1) AS name
```

- ❖ Using subqueries also has the advantage that SQL may be better able to optimize the entire query than it could if it had to materialize the first SELECT.

---

# Overview

---

- ❖ Intro to SQL
- ❖ Tables and schemas
- ❖ SQL queries - SELECT
- ❖ MySQL database management
  - Databases
  - Creating and populating tables
- ❖ Joins

# Joins

---

- ❖ We have mentioned that there are actually several flavors of joins. Recall the syntax of a join query:

```
SELECT expression [, expression ...]  
FROM table1 JOIN table2  
ON condition-involving-both-tables;
```

and its meaning: *For each pair of records from table1 and table2, test the condition; if it is true, concatenate the two records and put them into the resulting table.*

- ❖ The join condition is almost always a simple equality between values in two columns (one from each table); this is the *join column*. (In the earlier example, it was the “code” and “IllnessCode” columns from two tables.)



# Joins

---

- ❖ A question that arises with this definition is this: What happens if a record in *table1* doesn't match any records in *table2*?
  - The obvious thing is that the orphaned record in *table1* does not appear in the result. Similarly for a record in *table2* that doesn't match any records in *table1*.
- ❖ For cases where you prefer not to lose information by doing a join, there are versions of join that act differently.
  - In our example, we might want to remember all the illness descriptions in the joined table, even for illnesses that don't occur in the Admissions table.
- ❖ To retain unmatched records, we use an *outer join*.

## Joins

- ❖ For this discussion, we use slightly modified versions of the tables from yesterday.

Admissions		
<i>Facility</i>	<i>IllnessCode</i>	<i>Lengthofstay</i>
St. Marks	141	2
Mercy	152	4
Mercy	101	7
Jay County	152	5

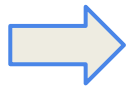
Illnesses	
<i>Code</i>	<i>Description</i>
94	Tonsillitis
141	Asthma
152	Broken arm
160	Appendicitis

- ❖ The difference is that we have an illness in Admissions that does not occur in Illnesses. We already had an illness in Illnesses (Appendicitis) for which there were no admissions.

## Joins

- ❖ Using the earlier join query, we get the following result. This is technically called an “inner join,” although it is often referred to as just “join.” This is slightly different from what we said earlier: In the result of a join, every column in both tables is included, *even the ones that are duplicates*. (We write INNER JOIN instead of JOIN to be explicit.)

```
SELECT * FROM Admissions INNER JOIN Illnesses  
ON Admissions.IllnessCode = Illnesses.Code;
```



<i>Facility</i>	<i>IllnessCode</i>	<i>Lengthofstay</i>	<i>Code</i>	<i>Description</i>
St. Marks	141	2	141	Asthma
Mercy	152	4	152	Broken arm
Jay County	152	5	152	Broken arm

## Outer joins

---


- ❖ In an outer join, we change the meaning of the join operation: *Do the same as in a regular join, but if a record from table1 is not matched by any record from table2, we act as if table2 includes a matching record, in which all columns are NULL, and similarly for records of table2 not matched by any record of table1.*
  - NULL is a special value used in relations to indicate that a cell is empty.
- ❖ To use an outer join instead of an inner join, add the words OUTER JOIN between the table name (instead of just a comma).

## Outer joins

- ❖ For the outer join in our example, pretend Admissions has a record with IllnessCode = 160 (and all other fields null), and Illnesses has a field with Code = 101.

```
SELECT * FROM Admissions FULL OUTER JOIN Illnesses
ON Admissions.IllnessCode = Illnesses.Code;
```

For illustrative purposes only. **This query is not valid in MySQL.**



<i>Facility</i>	<i>IllnessCode</i>	<i>Lengthofstay</i>	<i>Code</i>	<i>Description</i>
St. Marks	141	2	141	Asthma
Mercy	152	4	152	Broken arm
Jay County	152	5	152	Broken arm
Mercy	101	7	NULL	NULL
NULL	NULL	NULL	160	Appendicitis

## Left and right outer joins

- ❖ We may want to retain data only from table1 or table2, but not both. The outer join we just illustrated is sometimes called a “full” outer join. Left and right outer joins do the “obvious”:

```
SELECT * FROM Admissions LEFT OUTER JOIN Illnesses  
ON Admissions.IllnessCode = Illnesses.Code;
```



<i>Facility</i>	<i>IllnessCode</i>	<i>Lengthofstay</i>	<i>Code</i>	<i>Description</i>
St. Marks	141	2	141	Asthma
Mercy	152	4	152	Broken arm
Jay County	152	5	152	Broken arm
Mercy	101	7	NULL	NULL

## Left and right outer joins

---

```
SELECT * FROM Admissions RIGHT OUTER JOIN Illnesses  
ON Admissions.IllnessCode = Illnesses.Code;
```



<i>Facility</i>	<i>IllnessCode</i>	<i>Lengthofstay</i>	<i>Code</i>	<i>Description</i>
St. Marks	141	2	141	Asthma
Mercy	152	4	152	Broken arm
Jay County	152	5	152	Broken arm
NULL	NULL	NULL	160	Appendicitis

## Outer joins

---

- ❖ We noted earlier that the FULL OUTER JOIN query was not valid. MySQL actually does not implement full outer joins. However, you may have noted that the full outer join is nothing but the union of the left and right outer joins.
- ❖ MySQL implements the UNION operator:

```
select-command UNION select-command ;
```

- ❖ Implement the full outer join like this:

```
SELECT * FROM Admissions LEFT OUTER JOIN Illnesses  
ON Admissions.IllnessCode = Illnesses.Code  
UNION  
SELECT * FROM Admissions RIGHT OUTER JOIN Illnesses  
ON Admissions.IllnessCode = Illnesses.Code;
```



## Joins using subqueries

---

- ❖ It is quite common to do joins of tables with themselves or with tables derived from them.
- ❖ Example: Find all hospital stays whose *lengthofstay* matches the maximum *lengthofstay* overall. You need to find the maximum first (with a query) before you can find the stays that match it (with a join).

```
CREATE TABLE maxstay  
AS SELECT MAX(lengthofstay) AS maxstay FROM hospital;  
  
SELECT * FROM hospital INNER JOIN maxstay  
ON lengthofstay = maxstay;
```

## Joins using subqueries

---

- ❖ You can instead use the subquery syntax, i.e. include a SELECT statement in those places where we have been using a table name.

```
SELECT expression, ...  
FROM (SELECT ...) name1 INNER JOIN (SELECT ...) name2  
ON condition involving name1 and name2;
```

```
SELECT *  
FROM hospital  
INNER JOIN (  
    SELECT MAX(Lengthofstay) AS maxstay  
    FROM hospital  
    ) AS maxstaytable  
ON Lengthofstay = maxstaytable.maxstay;
```

## Exercise 8 - Joins

- ❖ Write queries to do the following between `hospital` and `diagcodes`:
  - `join`
  - `left outer join`
  - `right outer join`
  - `full outer join`
- ❖ You'll need to use the UNION trick to implement the FULL OUTER JOIN.

## Side Note: SQL Injections



## Exercise 2 solution

```
SELECT DISTINCT paymentsource FROM hospital;  
SELECT DiagnosisCode, Severity FROM hospital;  
SELECT DISTINCT DiagnosisCode, Severity FROM hospital;
```

## Exercise 3 solution

- a. 

```
SELECT *  
FROM hospital  
WHERE admissiontype = 'Urgent';
```
- b. 

```
SELECT *, totalcharges / lengthofstay AS chargeperday  
FROM hospital  
WHERE admissiontype = 'Urgent';
```
- c. 

```
SELECT DISTINCT age  
FROM hospital  
WHERE severity > 2;
```

## Exercise 4 solution

- a. 

```
SELECT COUNT(gender), gender
FROM hospital
GROUP BY gender;
```
- b. 

```
SELECT severity, AVG(lengthofstay)
FROM hospital
GROUP BY severity;
```
- c. 

```
SELECT severity, gender, AVG(totalcharges)
FROM hospital
GROUP BY severity, gender;
```

## Exercise 8 solution

```
SELECT * FROM hospital INNER JOIN DiagCodeMap  
ON hospital.DiagnosisCode = DiagCodeMap.DiagnosisCode;
```

```
SELECT * FROM hospital LEFT OUTER JOIN DiagCodeMap  
ON hospital.DiagnosisCode = DiagCodeMap.DiagnosisCode;
```

```
SELECT * FROM hospital RIGHT OUTER JOIN DiagCodeMap  
ON hospital.DiagnosisCode = DiagCodeMap.DiagnosisCode;
```

```
SELECT * FROM hospital LEFT OUTER JOIN DiagCodeMap  
ON hospital.DiagnosisCode = DiagCodeMap.DiagnosisCode  
UNION  
SELECT * FROM hospital RIGHT OUTER JOIN DiagCodeMap  
ON hospital.DiagnosisCode = DiagCodeMap.DiagnosisCode;
```