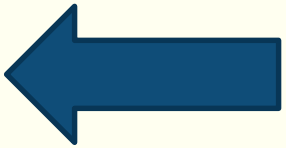
SQL Training on Genius

A Structured Query Language training on how to extract information from Genius SIS

SQL – Structured Query Language

- Used to work with a database
  - Query data 
  - Change data
  - Change data structures
    - –Tables
    - –Columns
    - –Views

SQL Training Scope

- Querying for Data (“SELECT”)
- Genius SIS database structures
  - Tables
  - Columns
  - Views

First example!

```
select FirstName, LastName
from Students
```

These are the columns in the table

This is a table

- SELECT command, followed by comma separated list of columns to show in result set
- FROM clause to specify which table(s) to get information from (get it?) ☺

FirstName	LastName
Paul	McCartney
Albert	Einstein
Sigmund	Freud
Ringo	Starr
David	Letterman
George	Harrison
James	Taylor

SELECT: Aliases and Special Ops

```
select LastName as Surname,
       Email as [Email Address],
       FirstName + ' ' + LastName [Full Name]
from Students
```

- "LastName" is aliased to "Surname"
- When the alias contains blank characters, use brackets
- When aliasing, you can use the **as** keyword to introduce the alias, but that's not required
- You can perform operations on columns to create a "new" column
- The **+** sign performs a concatenation operation on character fields

Surname	Email Address	Full Name
McCartney	paul@galileosis.com	Paul McCartney
Einstein	einstein@galileosis.com	Albert Einstein
Freud	freud@galileosis.com	Sigmund Freud
Starr	ringo@galileosis.com	Ringo Starr
Letterman	david@galileosis.com	David Letterman
Harrison	george@galileosis.com	George Harrison

Filtering Data (WHERE clause)

```
select LastName, Email [Email Address]
from Students
where FirstName = 'John'
```

- = operator allows to search for equality in column value
- Other operators are available:
  - = (equal to)
  - > (greater than)
  - >= (greater than or equal to)
  - < (less than)
  - <= (less than or equal to)
  - <> (different from)

```
select FirstName, LastName
from Students
where FirstName > 'John'
```

```
select FirstName, LastName, GradeLevel
from Students
where GradeLevel >= 10
```

```
select FirstName, LastName, GradeLevel, DOB
from Students
where DOB >= '1998-01-01'
```

Filtering Data: Special Clauses

```
select FirstName, LastName, Email [Email Address]
from Students
where FirstName like 'John%'
```

- **LIKE** operator allows to search for parts of a column value
- **%** character is a multiple characters wildcard so:
  - `FirstName like 'John%'` brings students whose first name starts with John
  - `FirstName like '%John'` brings students whose first name ends with John
  - `FirstName like '%John%'` brings students whose first name contains John

```
select FirstName, LastName, DOB
from Students
where DOB between '2000-08-01' and '2001-07-31'
```

- **BETWEEN** operator allows to search ranges in a column value
- Upper and lower boundaries values are included in the results

Filtering Data: Multiple Clauses

```
select FirstName, LastName
from Students
where FirstName = 'John'
or FirstName = 'George'
```

- **OR** operator allows to bring records where **either** condition is met

```
select FirstName, LastName
from Students
where FirstName = 'John'
and LastName = 'Lennon'
and DOB = '1998-03-20'
```

- **AND** operator allows to bring records where **all** conditions are met

Ordering Results (ORDER BY)

```
select FirstName, LastName
from Students
where GradeLevel = 11
order by LastName asc, FirstName
```

- **ORDER BY** clause allows to specify multiple fields for sorting the result
- By default, sorting is done in ascending order (from first/smallest to last/greatest), so the **asc** keyword is optional and can be omitted
- The sorting order is defined for each column in the **order by** clause. In the above example it reads as "order by last name in ascending order then by first name in ascending order"

```
select FirstName, LastName, GradeLevel
from Students
order by GradeLevel desc, LastName, FirstName
```

- **desc** keyword can be used to specify descending order (from last/greatest to first/smallest)

Limiting Results (TOP N)

```
select TOP 10 FirstName, LastName
from Students
where GradeLevel = 11
order by LastName, FirstName
```

- **TOP N** clause allows to specify the maximum number of rows to be returned in the result set
- Usually combined with **order by** clause
- What about getting the **bottom n**??

```
select TOP 10 FirstName, LastName
from Students
where GradeLevel = 11
order by LastName desc, FirstName desc
```

- There's no **bottom n** clause, but you can teak the sorting order to achieve this: sorting by descending order

Comments

```
select FirstName, LastName
from Students
where GradeLevel = 11
--and FirstName = 'John'
```

- The -- characters are used to comment a line.
- Comments are not considered when the query runs
- Comments are useful for:
  - Documenting
  - Testing assumptions or scenarios without having to delete code
- **/* */** delimiters can be used to create blocks of comment. Whatever is inside the delimiters is considered a comment

```
select FirstName, LastName
from Students
where GradeLevel = 11
/* and FirstName = 'John'
   and LastName like '%son%' */
order by LastName, FirstName
```

Aggregate Functions

- **Aggregate Functions** are used to count, sum, average (etc.) a result set

```
select count(*)
from Students
```

- **count** aggregate function counts occurrences

```
select count(*) EnrollmentsForStudent12345
from Enrollments
where StudentIndex = 12345
```

- Of course, you can use filters and aliases

Aggregate Functions

```
select count(*) EnrollmentCount,
       max(StartDate) [Lastest Start Date],
       min(StartDate) [Earliest Start Date],
       avg(CurrentGrade) [Average Grade],
       sum(TotalAssignments) [Total Assignments]
from Enrollments
where StudentIndex = 12345
```

- **Max** aggregate function returns the maximum value of a column in the result set
- **Min** aggregate function returns the minimum value of a column in the result set
- **Avg** aggregate function returns the average value of a column in the result set
- **Sum** aggregate function returns the sum of the values of a column in the result set

Aggregate Functions: Grouping

- All those stats are great, but we see them for an individual student only. How about a list of students with those stats?

```
select StudentIndex, count(*) EnrollmentCount,
       max(StartDate) [Lastest Start Date],
       min(StartDate) [Earliest Start Date],
       avg(CurrentGrade) [Average Grade],
       sum(TotalAssignments) [Total Assignments]
from Enrollments
where Status = 'ACTIVE'
group by StudentIndex
```

- **GROUP BY** clause allows to group aggregate results by one ore more key columns

Grouping: Filtering results

- What if we just need the results for students with more than 1 enrollment?
- **WHERE** clause cannot be used, because it operates on the original record context. That means **WHERE** allows to filter the records to be used in the aggregated result (i.e. operate the aggregation only in ACTIVE enrollments)

```
select StudentIndex, count(*) EnrollmentCount,
       max(StartDate) [Lastest Start Date]
from Enrollments
where Status = 'ACTIVE'
group by StudentIndex
having count(*) > 1
```

- **HAVING** clause allows to filter based on the aggregates output
- In SQL Server, **EnrollmentCount** cannot be referenced in the **HAVING** clause, aggregate functions must be used. Other database engines may allow that kind of referencing

No Duplicates in Results (DISTINCT)

- **DISTINCT** clause allows to remove duplicates in the result set
- Duplicates are determined by examining and comparing the whole row (all columns). If values in all columns are the same in 2 or more rows, only one version of the row will be displayed in the output

```
select distinct StudentIndex
from Enrollments
```

- In the query above, students with more than one enrollment will be displayed only once

```
select count(distinct StudentIndex)
from Enrollments
```

- **DISTINCT** can also be used within aggregate functions
- When in aggregate functions, only non-duplicate values will be considered in the output calculations. Examples:
  - COUNT: only different values will be counted (i.e. values: 2, 2, 3 → result: 2)
  - SUM: only different values will be summed (i.e. values: 2, 2, 3 → result: 5)

Working multiple Tables (Joining)

- List of students and their grades and dates in a specific section:

```
select StudentIndex, CurrentGrade, StartDate, EndDate
from Enrollments
where Status = 'ACTIVE'
and SectionIndex = 552 --AP Computer Science A
```

- The student data is stored in the **Students** table
- What we need is a way to include both tables (**Enrollments** and **Students**) in the same query
- For that, we also need to specify the rules on how the tables relate
- Finally, we also have to specify which columns from which tables we want in the result set

Working multiple Tables (Joining)

- List of students and their grades, revised:

```
select stu.FirstName, stu.LastName, CurrentGrade, StartDate, EndDate
from Enrollments as enr
inner join Students stu on enr.StudentIndex = stu.StudentIndex
where enr.Status = 'ACTIVE'
and enr.SectionIndex = 552 --AP Computer Science A
```

- We can reference multiple tables by using a JOIN clause
- SQL allows specifying aliases for the tables, which is encouraged
- When listing the columns in **SELECT** clause, they need to be qualified by referencing the table (or alias) they come from
- In the **JOIN** clause, the rules of the relation between the tables must be defined in the **ON** clause
- By design, in Genius you can relate two different tables by using a key column such as StudentIndex, SectionIndex, etc.
- ***SomeEntityIndex*** columns in Genius indicate the ID of an *entity* and it can be used to relate to other tables with the same column name
  - Example: any table with a **StudentIndex** column can be related to the **Students** table

Working multiple Tables (Joining)

- Examples:

```
select usr.Username, stu.FirstName, usr.LastName
from Students stu
inner join Users usr on stu.UserIndex = usr.UserIndex
```

```
select tea.LastName + ', ' + tea.FirstName Teacher, sec.Name Section
from Teacher tea
inner join Sections sec on tea.TeacherIndex = sec.TeacherIndex
where sec.Status = 'ACTIVE'
```

```
select stu.LastName Student, sec.Name Section,
       enr.CurrentGrade, tea.LastName Teacher
from Students stu
inner join Enrollments enr on stu.StudentIndex = enr.StudentIndex
inner join Sections sec on enr.SectionIndex = sec.SectionIndex
inner join Teacher tea on tea.TeacherIndex = sec.TeacherIndex
where enr.Status = 'ACTIVE'
```