Introduction to SQL

What is SQL?

- *De facto* language for querying Relational Databases
- Short for Structured Query Language
- 4th Generation language
  - Declarative, not Imperative
  - Tell it what to do, not how to do it

Declarative, Not Imperative

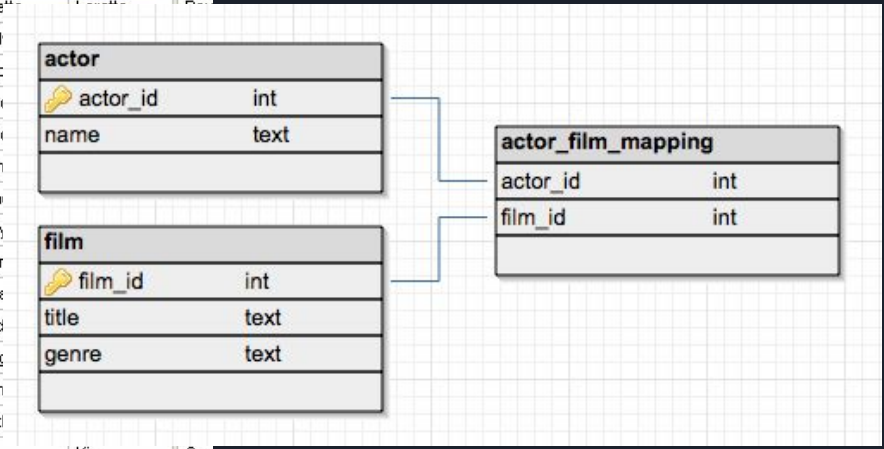
```
class Person {
    public string firstName;
    public string lastName;
}
//FROM
List<Person> inputRecords = new List();
List<string> outputSurnames = new List();
foreach (var person in inputRecords) {
    if (person.firstName == "Bob") { //WHERE
        outputSurnames.Add(person.lastName); //SELECT
    }
}
return outputSurnames;
```

```
SELECT lastName
FROM people
WHERE firstName = 'Bob'
```

What is an RDBMS?

- Acronym for Relational Database Management System
- Works with tables and the relationships between them
- SQL and RDBMS have been a good pairing for decades

	PersonID	Address	Address	City	City	FirstName	FirstName	Las
1	1	1040 East Street	1040 East Street	Plateau City	Plateau City	Lore		
2	2	154 Baltic Walk	154 Baltic Walk	Excelsior	Excelsior	Evel		
3	3	952 Tennessee Avenue	952 Tennessee Avenue	Embarcadero	Embarcadero	Harc		
4	4	780 Fourth Lane	780 Fourth Lane	Tenderloin	Tenderloin	Cha		
5	5	1079 Beach Way	1079 Beach Way	Cow Hollow	Cow Hollow	San		
6	6	758 North Lane	758 North Lane	North Beach	North Beach	Kath		
7	7	978 Eighth Walk	722 Arrow Lane	Miraloma Park	Columbus	Pam		
8	8	247 Fifth Place	247 Fifth Place	Western Addition	Western Addition	Emily		
9	9	843 States Street	843 States Street	Noe Valley	Noe Valley	Verr		
10	10	749 Washington Street	749 Washington Street	Civic Center	Civic Center	Willi		
11	11	360 Tennessee Place	360 Tennessee Place	Fisherman's Wharf	Fisherman's Wharf	Glad		
12	12	14 Oriental Place	14 Oriental Place	Buena Vista	Buena Vista	Marg		
13	13	668 Lower Avenue	668 Lower Avenue	Diamond Heights	Diamond Heights	Kath		
14	14	896 Third Street	896 Third Street	Civic Center	Civic Center	Bert		
15	15	229 Kentucky Place	229 Kentucky Place	Ocean View	Ocean View	Kim	Kim	Gra
16	16	1019 Marvin Gardens Place	1019 Marvin Gardens Place	Potrero Hill	Potrero Hill	Steve	Steve	Cur



How’s data stored in an RDBMS?

- Column: an attribute of our data we want to store. Columns can hold different types of data.
  - Numbers
  - Text
  - Dates
  - Boolean values (true/false)
- Row: an individual entry in a table. Holds information for some or all of the columns that table defines.
- Table: holds a collection of rows and columns
- Schema: contains all the information about the structure of your database.

Common RDBMS's

Our Sandbox

- https://data.stackexchange.com/stackoverflow/query/new

FROM Clause

- Selects which table(s) you want to query

```
FROM Posts
```

SELECT Clause

- Specifies what fields of a table you want returning
- Comma-separated list of field names

```
SELECT Id, PostTypeId, CreationDate, ViewCount, Title
```

- Or return all columns

```
SELECT *
```

SELECT Clause

- Can limit number of results

```
SELECT TOP 10 Id, PostTypeId,
CreationDate, ViewCount, Title
```

SELECT Clause

- Specifies what fields of a table you want returning
- Comma-separated list of field names

```
SELECT Id, PostTypeId, CreationDate, ViewCount, Title
```

- Or can return all columns

```
SELECT *
```

- Can limit number of results

```
SELECT TOP 10 Id, PostTypeId, CreationDate, ViewCount, Title
```

WHERE Clause

- Specify predicates that evaluate whether a particular record should be returned

```
WHERE PostTypeId = 1
    AND CreationDate >= '2016-07-01'
```

WHERE Clause

- Text Wildcard Matching

```
SELECT TOP 10
    Id, PostTypeId, CreationDate,
ViewCount, Title
FROM Posts
WHERE Title LIKE '%help%'
```

```
SELECT TOP 10
    Id, PostTypeId, CreationDate,
ViewCount, Title
FROM Posts
WHERE Title LIKE '%help%java%'
```

ORDER BY Clause

Sort your rows specifying one or more columns as the sorting criteria.

```
ORDER BY CreationDate
```

Set sort direction with DESC and ASC (the default)

```
ORDER BY CreationDate DESC
```

Can add further columns as sorting criteria to disambiguate when two rows have the same column value

```
ORDER BY CreationDate DESC, OwnerUserId
```

Putting It All Together

```
SELECT
    Id, Title, CreationDate,
    OwnerUserId
FROM Posts
WHERE CreationDate >= '2016-07-01'
ORDER BY CreationDate DESC
```

Modifying Data

- INSERT

```
INSERT INTO Users (Id, DisplayName, Location) VALUES (1, 'Roberto',
'Milwaukee')
```

- UPDATE

```
UPDATE Users
SET Reputation = 0
WHERE AboutMe LIKE '%expert%'
```

- DELETE

```
DELETE FROM Users WHERE AboutMe LIKE '%expert%'
```

JOINS

- Join related data together

```
SELECT TOP 100
    p.CreationDate, p.Title,
p.LastActivityDate, i.DisplayName, i.AboutMe,
i.UpVotes, i.DownVotes
FROM Posts p
    JOIN Users i ON p.OwnerUserId = i.Id
```

JOINS

- Can join multiple tables

```
SELECT TOP 100
    *
FROM Posts p
    JOIN Users i ON p.OwnerUserId = i.Id
    JOIN Comments c ON p.Id = c.PostId
```

JOINS

- But be careful when you do…!

```
SELECT TOP 100
    *
FROM Posts p
    JOIN Users i ON p.OwnerUserId = i.Id
    JOIN Comments c ON p.Id = c.PostId
WHERE p.Id = 6470651
```

JOINS

- Join related data together

```
SELECT TOP 100
   p.CreationDate, p.Title, p.LastActivityDate, i.DisplayName, i.AboutMe, i.UpVotes,
i.DownVotes
FROM Posts p
   JOIN Users i ON p.OwnerUserId = i.Id
```

- Can join multiple tables

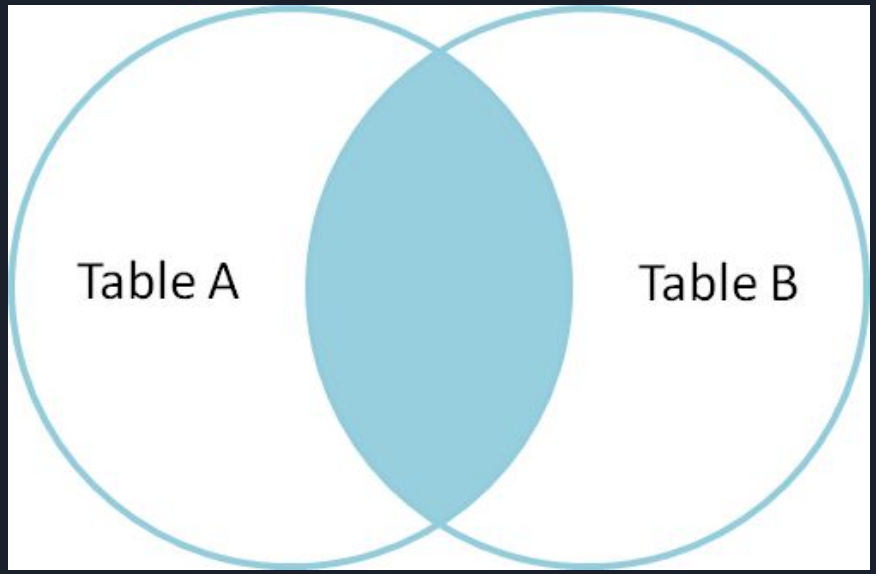
```
SELECT TOP 100
   *
FROM Posts p
   JOIN Users i ON p.OwnerUserId = i.Id
   JOIN Comments c ON p.Id = c.PostId
```

- But be careful when you do…!

```
SELECT TOP 100
   *
FROM Posts p
   JOIN Users i ON p.OwnerUserId = i.Id
   JOIN Comments c ON p.Id = c.PostId
WHERE p.Id = 6470651
```

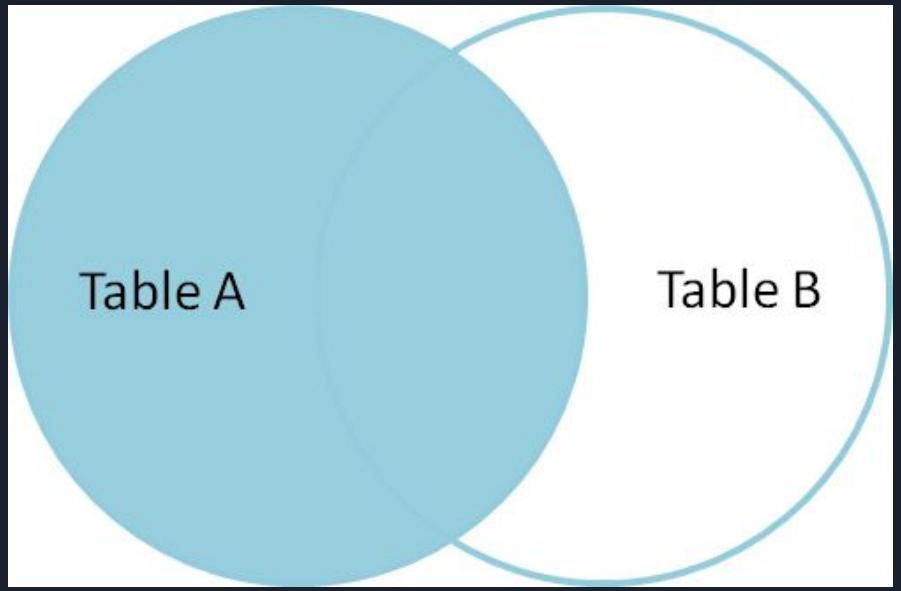
JOINS

- INNER Join
  - Rows on the adjoining table have to match the condition, otherwise the whole row is discounted
  - Matching records in both tables



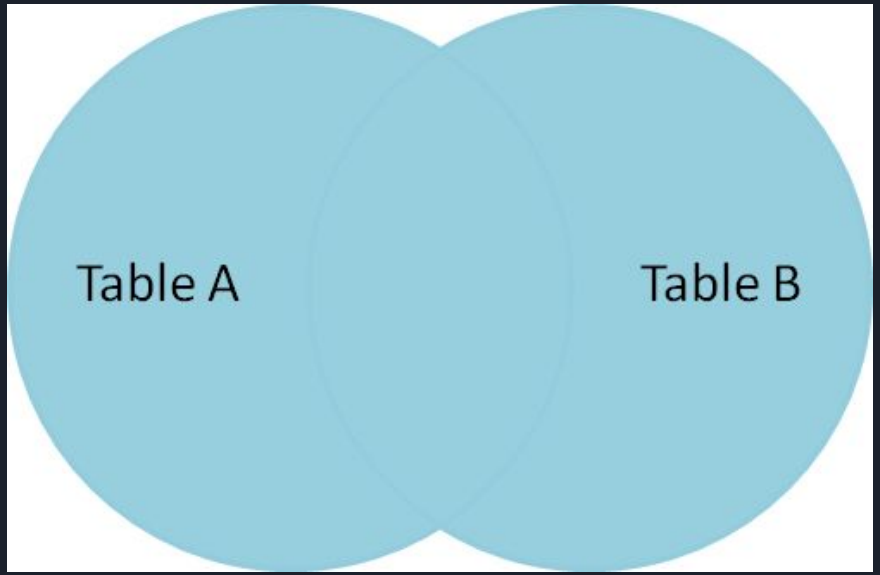
JOINS

- LEFT/RIGHT OUTER Join
  - If no rows on the adjoining table match join condition, only data from base table is returned in row



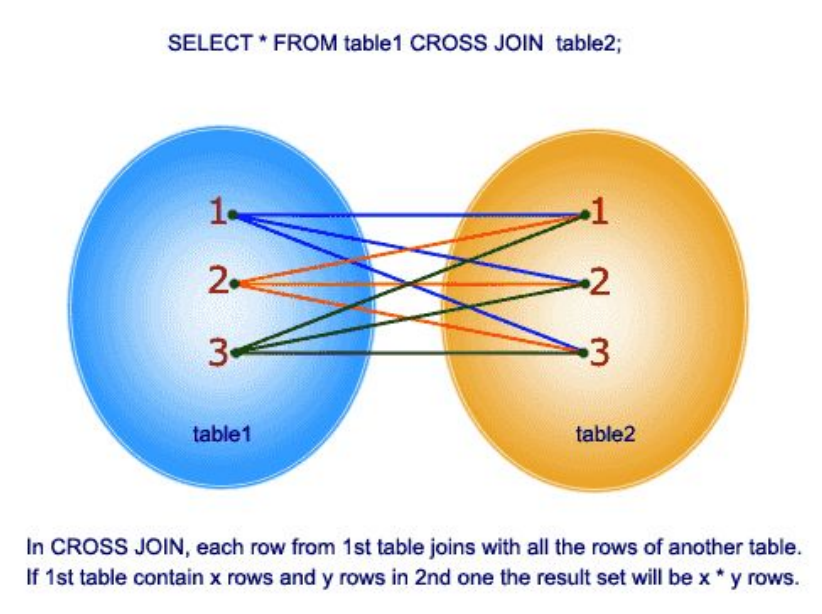
JOINS

- FULL OUTER Join
  - Return rows from both tables, regardless if they match or not



JOINS

- CROSS Join
  - Cartesian Product
  - Can be restricted using ON or WHERE clause
  - Can Cross Join multiple tables



Still with me?

Aliasing

Rename selected columns or tables

```
SELECT
    p.Id AS UserId,
    p.PostTypeId AS StackOverflowPostTypeId
FROM Posts AS p
```

Functions

Pass a column as an input, do something useful with it, return a result back to your query

```
SELECT TOP 100
    Id,
    PostTypeId,
    MONTH(CreationDate) AS MonthCreated,
    YEAR(CreationDate) AS YearCreated,
    SUBSTRING(Title, 1, 10) AS ShortTitle
```

WHERE Clause

- Can contain complex nested conditions and functions

```
WHERE PostTypeId = 1
    AND (CreationDate >= '2016-07-01'
OR MONTH(CreationDate) = 4)
```

WHERE Clause

- Specify condition that evaluate whether a particular record should be returned

```
WHERE PostTypeId = 1
   AND CreationDate >= '2016-07-01'
```

- Can contain complex nested conditions and functions

```
WHERE PostTypeId = 1
   AND (CreationDate >= '2016-07-01' OR
MONTH(CreationDate) = 4)
```

Aggregations

- Allows applying mathematical operations over a data set
- Works in conjunction with GROUP BY
- Specify what columns to group by
- Then what operations to perform on the grouped data

```
SELECT TOP 100
    p.Id AS PostId, COUNT(c.Id) AS CommentCount
FROM Posts p
    JOIN Comments c ON p.Id = c.PostId
GROUP BY p.id
```

Aggregations

- Other common functions are AVG, MIN, MAX, SUM

```
SELECT TOP 100
    u.Id, u.DisplayName,
    SUM(c.Score) AS TotalScore,
    AVG(c.Score) AS AverageScore,
    MIN(c.Score) AS MinScore,
    MAX(c.Score) AS MaxScore
FROM Comments c
    INNER JOIN Users u on c.UserId = u.Id
GROUP BY u.Id, u.DisplayName
```

DML vs DDL

- Data Manipulation Language
  - Everything we've done up to now...
- Data Definition Language
  - SQL that defines how to create tables

```sql
CREATE TABLE Persons (
    PersonID int,
    LastName varchar(255),
    FirstName varchar(255),
    Address varchar(255),
    City varchar(255)
);
```

DML vs DDL

- And ALTER them...

```
ALTER TABLE Persons ADD
DateOfBirth DATETIME;
```

- And DROP them...

```
DROP TABLE
Persons;
```

Aggregations

- HAVING allows to filter post-aggregation
  - WHERE filters pre-aggregation

```
SELECT TOP 10
    p.Id AS PostId, COUNT(c.Id) AS CommentCount
FROM Posts p
    JOIN Comments c ON p.Id = c.PostId
WHERE Title LIKE '%python%'
GROUP BY p.id
HAVING COUNT(c.Id) > 20
```

Subqueries

- Allow nesting of queries
- Can compose queries on top of each other
- Useful for queries with intermediate steps

```
SELECT topScores.TotalScore, u.Id, u.DisplayName, u.Location
FROM Users u
    INNER JOIN ( SELECT TOP 10
                    c.UserId AS UserId, SUM(c.Score) AS TotalScore
                FROM Comments c
                WHERE c.UserId IS NOT NULL
                GROUP BY c.UserId
                ORDER BY SUM(c.Score) DESC
) topScores ON u.Id = topScores.UserId
ORDER BY topScores.TotalScore DESC
```

Advanced Topics

- (Recursive) Common Table Expressions
- Windowing Functions
- Indexing
- User-Defined Functions

What's the future?

- End of RDBMS?
  - NoSQL Alternatives - different types of DBMS
    - Columnar DBs
    - Key-Value DBs
    - Graph DBs
    - etc.
- SQL is still the go-to query language for many of these

References

- https://www.w3schools.com/sql/default.asp
- http://db-engines.com/
- https://use-the-index-luke.com/
- https://blog.codinghorror.com/a-visual-explanation-of-sql-joins/

Try it Yourself!

- https://www.postgresql.org/download/
- https://dev.mysql.com/downloads/

Questions?

Tell us what you think on https://codeupleeds.github.io/feedback or scan the QR code below!