Databases

COEN 161

What are Databases?

- Until now, we have been keeping our data in memory, primarily in global variables
- This works for small sets of data but larger applications with large amounts of users may be processing more data
- Keeping data in memory can also make some operations slower, such as searching, deleting adding, etc.
- If the server crashes, all your data will be lost
- Databases address these problems and many more

What are Database?

- Databases are a structured collection of data
- The structure allows someone to query the data efficiently
- It also makes adding and deleting data easier
- Databases are nothing more than files, so they are persisted even if the application accessing the database stops running
- Some databases also add safety measures, like redundancy, to prevent data loss

Using Databases

- To build a web application that uses a database, you will need…
    - An RDBMS database program (i.e. MS Access, SQL Server, MySQL)
    - To use a server-side scripting language, like Node.js
    - To use SQL to get the data you want
    - To use HTML / CSS to style the page

Relational Database Management System

- An RDBMS, as well as other DBMSs, is a computer software program that interfaces with the database as well as the application or even a database user directly
- The RDBMS maintains the relations between the data that makes a database more efficient at maintaining data than other solutions
- This relation is established through the structure in which the data is kept
- Most RDBMSs store data in database objects called ***tables***
- A table is a collection of *related* data entries and consists of **columns and rows**
- Examples of RDMSs include Oracle, MySQL, and Microsoft Access

Structured Query Language

- The RDBMS program is the basis for SQL
- SQL lets you access and manipulate databases through the RDBMS
- The RDBMS understands SQL queries and uses them as instructions for operations on the database
- For example, this SQL query selects all the data from a table called *Customers*

```
SELECT * FROM Customers;
```

Structure Query Language

- Things that SQL can do
    - SQL can execute queries against a database
    - SQL can retrieve data from a database
    - SQL can insert records in a database
    - SQL can update records in a database
    - SQL can delete records from a database
    - SQL can create new databases
    - SQL can create new tables in a database
    - SQL can create stored procedures in a database
    - SQL can create views in a database
    - SQL can set permissions on tables, procedures, and views

Database Tables

- A database will often contain one or more tables
- Tables contain records (rows) with data
- This is the *customers* table and it contains five records

CustomerID	CustomerName	ContactName	Address	City	PostalCode	Country
1	Alfreds Futterkiste	Maria Anders	Obere Str. 57	Berlin	12209	Germany
2	Ana Trujillo Emparedados y helados	Ana Trujillo	Avda. de la Constitución 2222	México D.F.	05021	Mexico
3	Antonio Moreno Taquería	Antonio Moreno	Mataderos 2312	México D.F.	05023	Mexico
4	Around the Horn	Thomas Hardy	120 Hanover Sq.	London	WA1 1DP	UK

SQL Statements

- Most actions that need to be performed on a database can be done using an SQL statement

```
SELECT * FROM Customers;
```

- SQL keywords are NOT case-sensitive, but generally they are written in all-caps to differentiate between the keywords and the database/table/column names
- Depending on the database system, a semicolon may be required at the end of each statement
- Systems that use a semicolon allow multiple statements to be executed at once, each separated by a semicolon

SQL Commands

- The major SQL Commands are…
  - CREATE DATABASE - creates a new database
  - ALTER DATABASE - modifies a database
  - CREATE TABLE - creates a new table
  - ALTER TABLE - modifies a table
  - DROP TABLE - deletes a table
  - SELECT - extracts data from a database
  - UPDATE - updates data in a database
  - DELETE - deletes data from a database
  - INSERT INTO - inserts new data into a database

SQL Create Database

- To create a database, execute the following statement in the RDBMS

```
CREATE DATABASE databasename;
```

- Note: You must have admin privileges to create databases. In the ECC, a database is created for you.
- To see the list of databases you can use this statement

```
SHOW DATABASES;
```

SQL Delete Database

- To delete a database, execute the following statement in the RDBMS

```
DROP DATABASE databasename;
```

- Note: You must have admin privileges to delete databases.
- Once the database is deleted, you can see the list of databases you can use this statement

```
SHOW DATABASES;
```

SQL Create Table

- The CREATE TABLE statement is used to create a table within a database

```
CREATE TABLE table_name (
    column1 datatype,
    column2 datatype,
    column3 datatype,
   ....
);
```

- The column parameters specify the column names
- The data types can be supported data types including varchar, integer, and date

SQL Create Table

- The following example creates a table called "Persons" that contains five columns: PersonID, LastName, FirstName, Address, and City
- The PersonID column is of type int and will hold an integer.
- The LastName, FirstName, Address, and City columns are of type varchar and will hold characters, and the maximum length for these fields is 255 characters

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

SQL Delete Table

- The following statement deletes a table from the database

```
DROP TABLE table_name;
```

- Once deleted, the table and all its data doesn’t exist
- If you want to remove the data but not the table use this statement

```
TRUNCATE TABLE table_name;
```

SQL Alter Table

- The ALTER TABLE statement is used to add, delete, or modify columns in an existing table
- To add a column use this statement

```
ALTER TABLE table_name

ADD column_name datatype;
```

- To delete a column use this statement

```
ALTER TABLE table_name

DROP COLUMN column_name;
```

SQL Alter Table

- To modify a table, use the following statement

```
ALTER TABLE table_name

MODIFY COLUMN column_name datatype;
```

SQL Alter Table

- Example - The Persons Table

ID	LastName	FirstName	Address	City
1	Hansen	Ola	Timoteivn 10	Sandnes
2	Svendson	Tove	Borgvn 23	Sandnes
3	Pettersen	Kari	Storgt 20	Stavanger

- Now we add a column called DateOfBirth to the table

```
ALTER TABLE Persons

ADD DateOfBirth date;
```

SQL Alter Table

- The Persons table now looks like this

ID	LastName	FirstName	Address	City	DateOfBirth
1	Hansen	Ola	Timoteivn 10	Sandnes	
2	Svendson	Tove	Borgvn 23	Sandnes	
3	Pettersen	Kari	Storgt 20	Stavanger	

- Now we’re modifying the datatype of the DateOfBirth column

```
ALTER TABLE Persons

ALTER COLUMN DateOfBirth year;
```

SQL Alter Table

- If we aren’t happy with this column, we can drop it

```
ALTER TABLE Persons

DROP COLUMN DateOfBirth;
```

- Now our table looks like it first did

ID	LastName	FirstName	Address	City
1	Hansen	Ola	Timoteivn 10	Sandnes
2	Svendson	Tove	Borgvn 23	Sandnes
3	Pettersen	Kari	Storgt 20	Stavanger

SQL Constraints

- Constraints are special rules that can be applied to table columns
- Constraints can be specified when the table is created with the CREATE TABLE statement
- They can also be specified after the table is created with the ALTER TABLE statement

```
CREATE TABLE table_name (
    column1 datatype constraint,
    column2 datatype constraint,
    column3 datatype constraint,
    ....
);
```

SQL Constraints

- Some common constraints include
  - NOT NULL - Ensures that a column cannot have a NULL value
  - UNIQUE - Ensures that all values in a column are different
  - PRIMARY KEY - A combination of a NOT NULL and UNIQUE. Uniquely identifies each row in a table
  - FOREIGN KEY - Uniquely identifies a row/record in another table
  - DEFAULT - Sets a default value for a column when no value is specified

SQL NOT NULL

- To make a column not nullable, add the constraint when creating or altering the table

```
CREATE TABLE Persons (
    ID int NOT NULL,
    LastName varchar(255) NOT NULL,
    FirstName varchar(255) NOT NULL,
    Age int
);
```

- Note: If the table has already been created, you can add a NOT NULL constraint to a column with the ALTER TABLE statement.

SQL UNIQUE

- The UNIQUE constraint ensures that only unique values can be entered for a column

```
CREATE TABLE Persons (
    ID int NOT NULL,
    LastName varchar(255) NOT NULL,
    FirstName varchar(255),
    Age int,
    UNIQUE (ID)
);
```

SQL UNIQUE

- You can also apply the constraint to multiple columns using the following syntax

```
CREATE TABLE Persons (
    ID int NOT NULL,
    LastName varchar(255) NOT NULL,
    FirstName varchar(255),
    Age int,
    CONSTRAINT UC_Person UNIQUE (ID,LastName)
);
```

SQL PRIMARY KEY

- The PRIMARY KEY constraint uniquely identifies each record in a database table
- Primary keys must contain UNIQUE values, and cannot contain NULL values
- A table can have only one primary key, which may consist of single or multiple fields.

```
CREATE TABLE Persons (
    ID int NOT NULL,
    LastName varchar(255) NOT NULL,
    FirstName varchar(255),
    Age int,
    PRIMARY KEY (ID)
);
```

SQL PRIMARY KEY

- To create a PRIMARY KEY constraint on the "ID" column when the table is already created, use the following statement

  ```
  ALTER TABLE Persons

  ADD PRIMARY KEY (ID);
  ```

- To name a PRIMARY KEY constraint, and for defining a PRIMARY KEY constraint on multiple columns, use the following statement

  ```
  ALTER TABLE Persons

  ADD CONSTRAINT PK_Person PRIMARY KEY (ID,LastName);
  ```

SQL PRIMARY KEY

- Note: If you use the ALTER TABLE statement to add a primary key, the primary key column(s) must have the NOT NULL constraint
- To DROP a primary key, use the following statement

```
ALTER TABLE Persons

DROP PRIMARY KEY;
```

SQL AUTO INCREMENT

- Auto-increment allows a unique number to be generated automatically when a new record is inserted into a table
- Often this is the primary key field that we would like to be created automatically every time a new record is inserted
- The following creates a table with an auto increment ID field

```
CREATE TABLE Persons (
    ID int NOT NULL AUTO_INCREMENT,
    LastName varchar(255) NOT NULL,
    FirstName varchar(255),
    Age int,
    PRIMARY KEY (ID)
);
```

SQL AUTO INCREMENT

- By default, the starting value for AUTO_INCREMENT is 1, and it will increment by 1 for each new record
- To set a different increment interval, use the following statement

```
ALTER TABLE Persons AUTO_INCREMENT=100;
```

- You can also set the interval when creating the table

SQL SELECT

- The SELECT statement is used to select data from a database
- The data returned is stored in a result table, called the result-set

```
SELECT column1, column2, ...

FROM table_name;
```

- Here, column1, column2, ... are the column names of the table you want to select data from
- If you want to select all the fields available in the table, use the following syntax

```
SELECT * FROM table_name;
```

SQL SELECT DISTINCT

- The SELECT DISTINCT statement is used to return only distinct (different) values
- Inside a table, a column often contains many duplicate values and sometimes you only want to list the different (distinct) values

```
SELECT DISTINCT column1, column2, ...

FROM table_name;
```

SQL WHERE clause

- The WHERE clause is used to filter records
- The WHERE clause is used to extract only those records that meet a specified condition

```
SELECT column1, column2, ...

FROM table_name

WHERE condition;
```

- Note: The WHERE clause is not only used in SELECT statement, it is also used in UPDATE, DELETE statement, etc.

SQL WHERE Clause

CustomerID	CustomerName	Address	City	PostalCode	Country
1	Alfreds Futterkiste	Obere Str. 57	Berlin	12209	Germany
2	Ana Trujillo Emparedados y helados	Avda. de la Constitución 2222	México D.F.	05021	Mexico
3	Antonio Moreno Taquería	Mataderos 2312	México D.F.	05023	Mexico
4	Around the Horn	120 Hanover Sq.	London	WA1 1DP	UK

- Given this table, we want to select all records where the country is Mexico

```
SELECT * FROM Customers
WHERE Country='Mexico';
```

SQL WHERE Clause

- SQL requires single quotes around text values
- However, numeric fields should not be enclosed in quotes

```
SELECT * FROM Customers

WHERE CustomerID=1;
```

SQL WHERE Clause

- The following operators can be used in the WHERE clause

Operator	**Description**
=	Equal
<>	Not equal. **Note:** In some versions of SQL this operator may be written as !=
>	Greater than
<	Less than
>=	Greater than or equal
<=	Less than or equal
LIKE	Search for a pattern
IN	To specify multiple possible values for a column

SQL AND, OR and NOT Operators

- The WHERE clause can be combined with AND, OR, and NOT operators
- The AND and OR operators are used to filter records based on more than one condition
- The AND operator displays a record if all the conditions separated by AND is TRUE
- The OR operator displays a record if any of the conditions separated by OR is TRUE
- The NOT operator displays a record if the condition(s) is NOT TRUE

```
SELECT * FROM Customers

WHERE NOT Country='Germany' AND (City='Berlin' OR City='München');
```

SQL ORDER BY

- The ORDER BY keyword is used to sort the result-set in ascending or descending order
- The ORDER BY keyword sorts the records in ascending order by default
- To sort the records in descending order, use the DESC keyword.

```
SELECT column1, column2, ...

FROM table_name

ORDER BY column1, column2, ... ASC|DESC
```

SQL INSERT INTO

- The INSERT INTO statement is used to insert new records in a table
- It is possible to write the INSERT INTO statement in two ways
- The first way specifies both the column names and the values to be inserted

```
INSERT INTO table_name (column1, column2, column3, ...)

VALUES (value1, value2, value3, ...);
```

SQL INSERT INTO

- If you are adding values for all the columns of the table, you do not need to specify the column names
- However, make sure the order of the values is in the same order as the columns in the table

```
INSERT INTO table_name

VALUES (value1, value2, value3, ...);
```

SQL INSERT INTO

CustomerID	CustomerName	ContactName	Address	City	PostalCode	Country
89	White Clover Markets	Karl Jablonski	305 - 14th Ave. S. Suite 3B	Seattle	98128	USA
90	Wilman Kala	Matti Karttunen	Keskuskatu 45	Helsinki	21240	Finland
91	Wolski	Zbyszek	ul. Filtrowa 68	Walla	01-012	Poland

- The following SQL statement inserts a new record in the "Customers" table

```
INSERT INTO Customers (CustomerName, ContactName, Address, City, PostalCode,
Country)

VALUES ('Cardinal', 'Tom B. Erichsen', 'Skagen 21', 'Stavanger', '4006',
'Norway');
```

SQL INSERT INTO

CustomerID	CustomerName	ContactName	Address	City	PostalCode	Country
89	White Clover Markets	Karl Jablonski	305 - 14th Ave. S. Suite 3B	Seattle	98128	USA
90	Wilman Kala	Matti Karttunen	Keskuskatu 45	Helsinki	21240	Finland
91	Wolski	Zbyszek	ul. Filtrowa 68	Walla	01-012	Poland
92	Cardinal	Tom B. Erichsen	Skagen 21	Stavanger	4006	Norway

- Notice that we did not insert any number into the CustomerID field
- The CustomerID column is an auto-increment field and will be generated automatically when a new record is inserted into the table

SQL UPDATE

- The UPDATE statement is used to modify the existing records in a table
- The WHERE clause specifies which record(s) that should be updated
- If you omit the WHERE clause, all records in the table will be updated

```
UPDATE table_name

SET column1 = value1, column2 = value2, ...

WHERE condition;
```

SQL DELETE

- The DELETE statement is used to delete existing records in a table
- The WHERE clause specifies which record(s) that should be deleted
- If you omit the WHERE clause, all records in the table will be deleted

```
DELETE FROM table_name

WHERE condition;
```

SQL COUNT()

- The COUNT() function returns the number of rows that matches a specified criteria

```
SELECT COUNT(column_name)

FROM table_name

WHERE condition;
```

SQL LIKE Operator

- The LIKE operator is used in a WHERE clause to search for a specified pattern in a column
- There are two wildcards used in conjunction with the LIKE operator
  - % - The percent sign represents zero, one, or multiple characters
  - _ - The underscore represents a single character

```
SELECT column1, column2, ...

FROM table_name

WHERE columnN LIKE pattern;
```

SQL LIKE

LIKE Operator	**Description**
WHERE CustomerName LIKE 'a%'	Finds any values that starts with "a"
WHERE CustomerName LIKE '%a'	Finds any values that ends with "a"
WHERE CustomerName LIKE '%or%'	Finds any values that have "or" in any position
WHERE CustomerName LIKE '_r%'	Finds any values that have "r" in the second position
WHERE CustomerName LIKE 'a_%_%'	Finds any values that starts with "a" and are at least 3 characters in length
WHERE ContactName LIKE 'a%o'	Finds any values that starts with "a" and ends with "o"

Resources

https://www.w3schools.com/sql/default.asp

https://www.w3schools.com/sql/sql_datatypes.asp