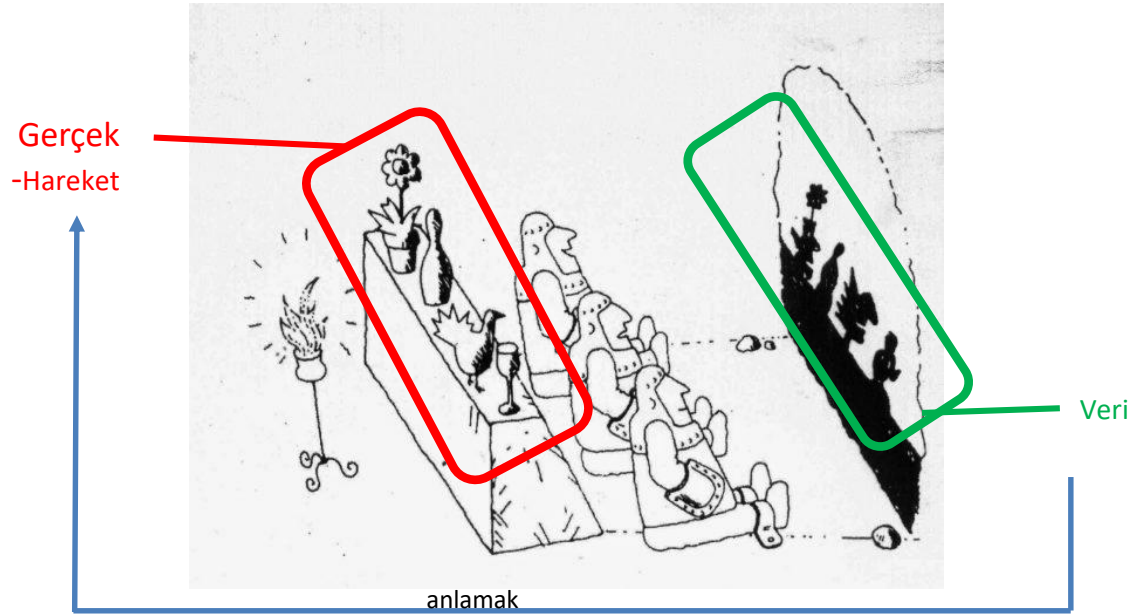


Data Management for Data Science

Data & Databases

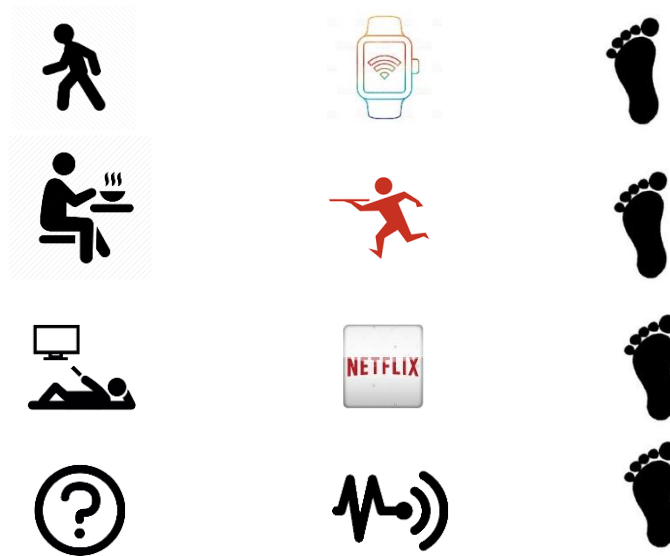
Başar Öztayşı
oztaysib@itu.edu.tr

Plato'nun Mağara Benzetmesi



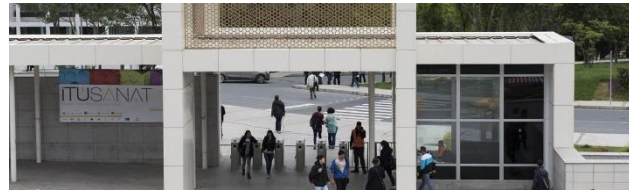
Life Cycle of Data

The Birth of Data



Life Cycle of Data

The Transmission of Data



Your Card Id



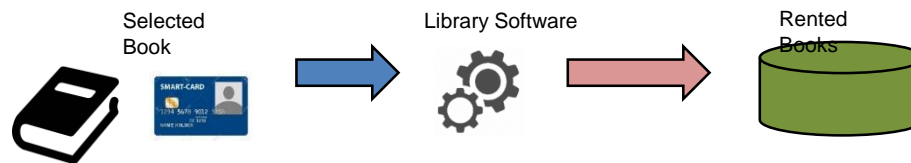
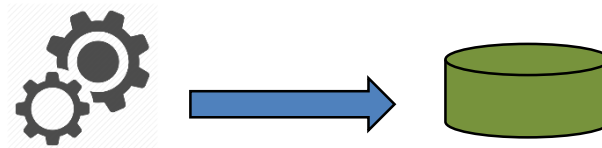
Valid Card



Response
GO / NO GO

Life Cycle of Data

The Storage of Data



Life Cycle of Data

The Analysis of Data



- Reports
- Predictions
- Prescriptions



Rented Books,
User Ids, Dates



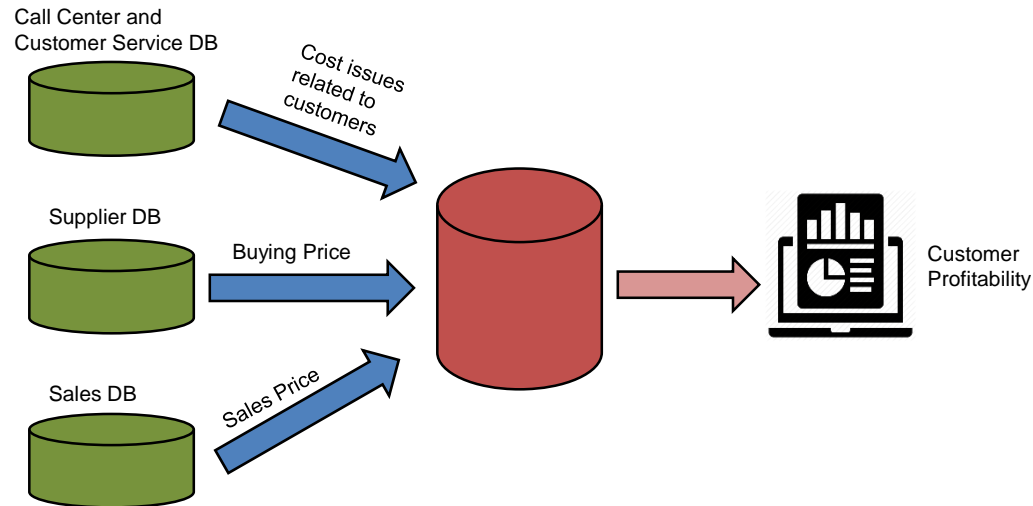
Analysis &
Algorithms



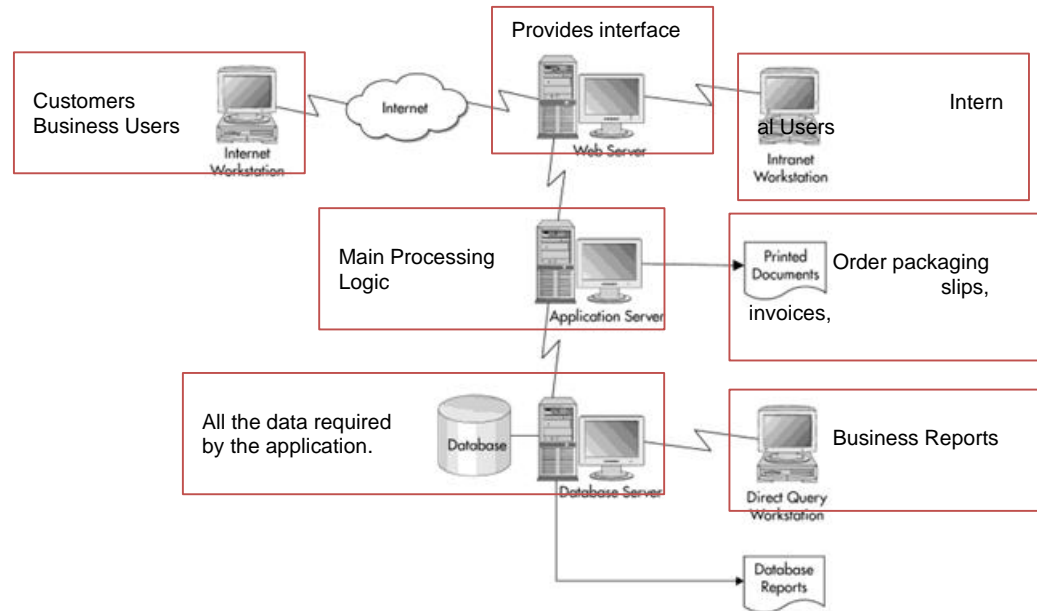
Most Popular Book
Most Active Student

Life Cycle of Data

The Aggregation of Data



A web-based IS



Web Server

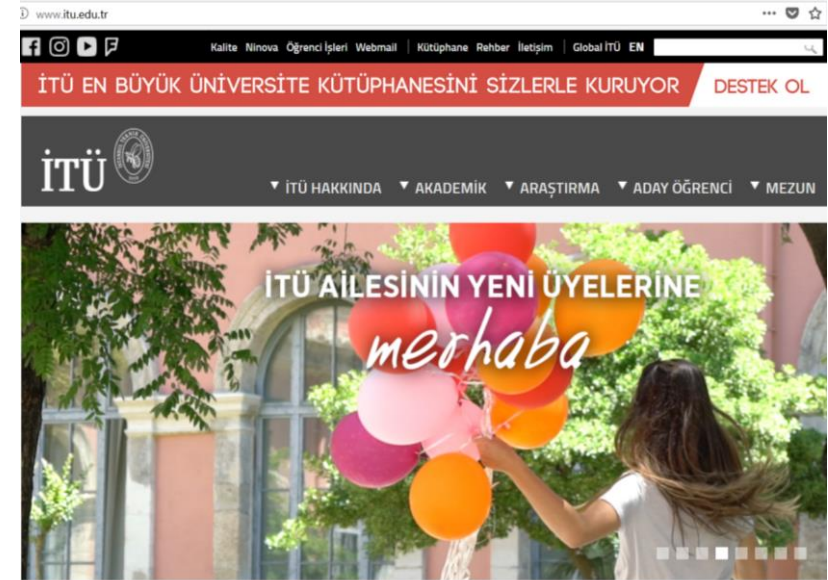
- When you browse www.itu.edu.tr
- Web Server:
 - HTML
 - Javascript / VB Script
 - CSS
 - AJAX
 - JSON
 - PHP
 - ASP.NET

```
<html>
<body>

<h1>My First Heading</h1>

<p>My first paragraph.</p>

</body>
</html>
```



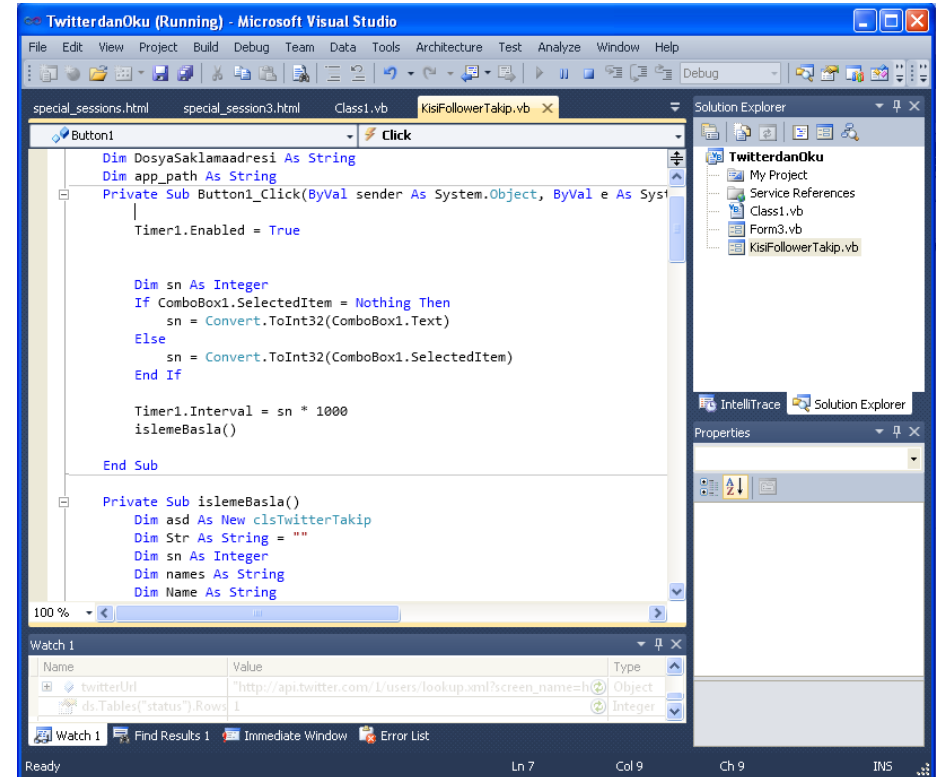
My First Heading

My first paragraph.

Application server

Application Server: The business logic Programming Languages

- VB.NET
- C#
- Java
- Delphi
- C



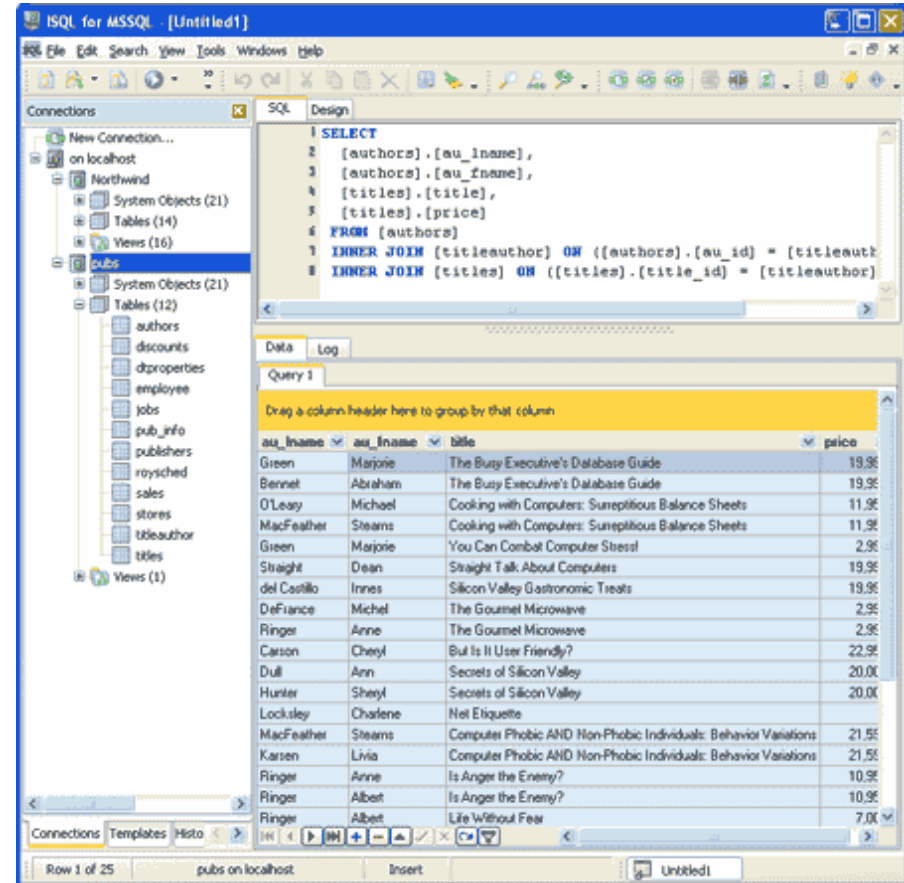
Database Server

Database Server:

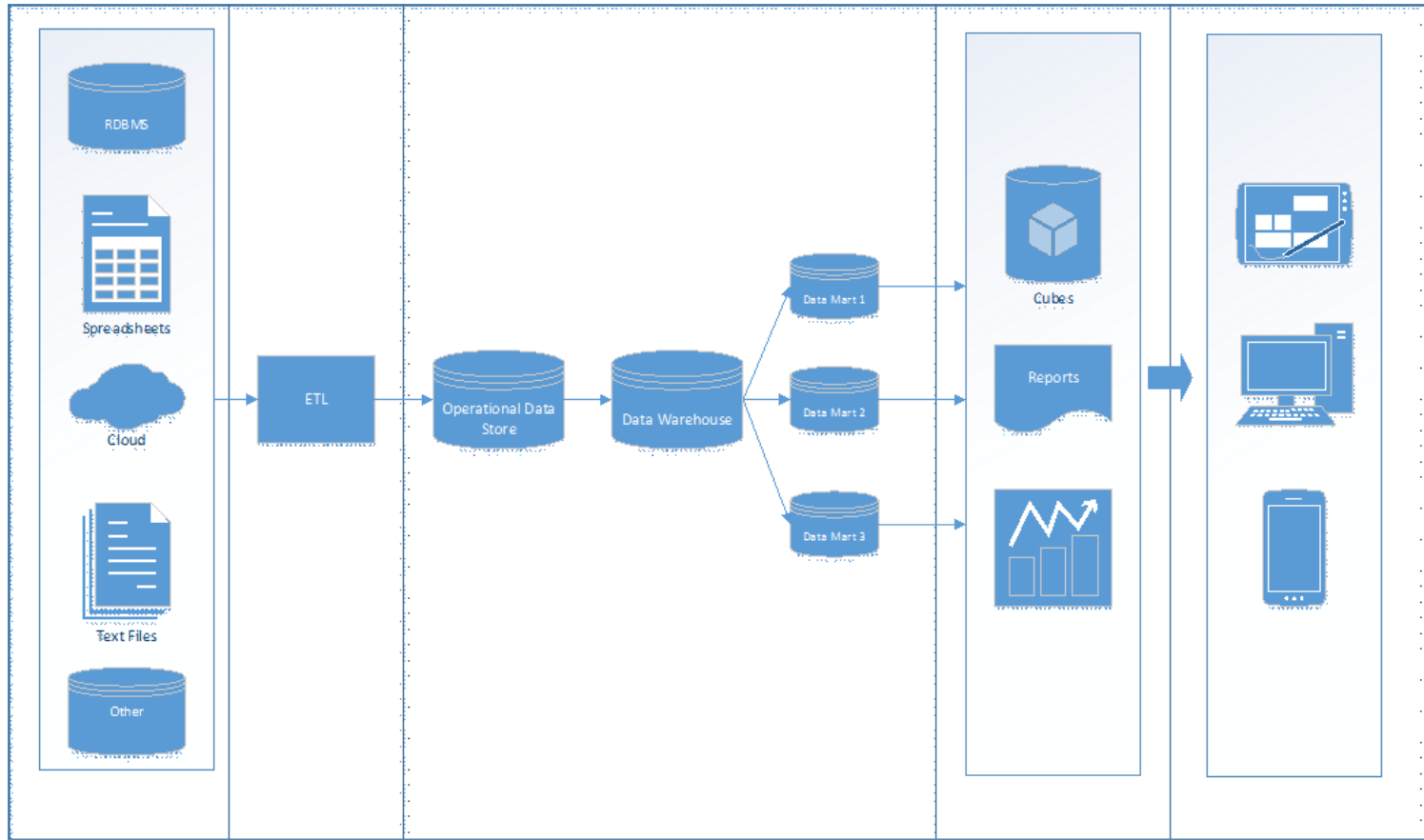
- MS SQL
- MySQL
- Oracle
- MS Access

SQL Language

- Select
- From
- Where...



Dataware House

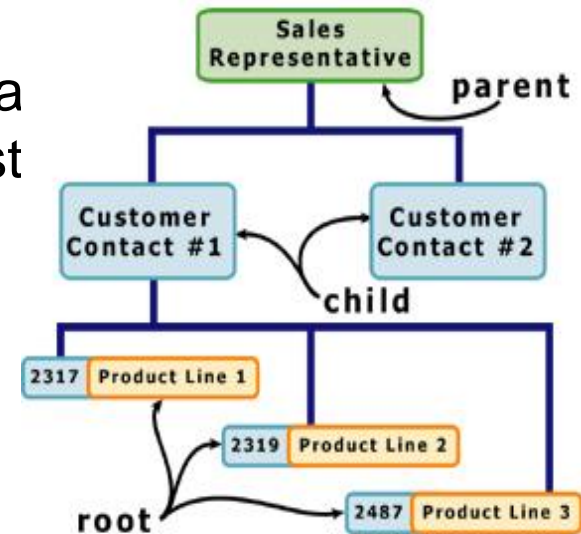


Types of Database Models



Hierarchical Model

- Oldest database models (1950). The hierarchical model assumes that a tree structure is the most frequently occurring relationship.
- This mislead restrictive view of relationships.
- The disadvantage of this type of database structure is that each child in the tree may have only one parent, and relationships or linkages between children are not permitted, even if they make sense from a logical standpoint.



Network Model

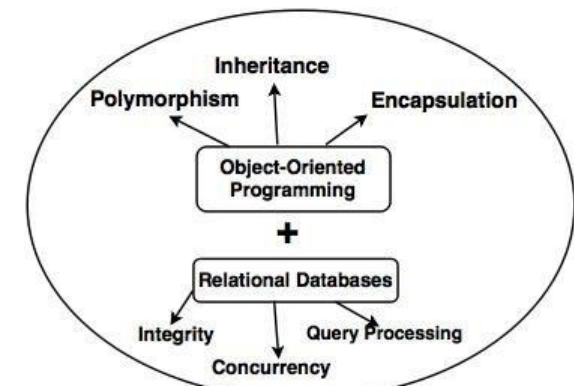
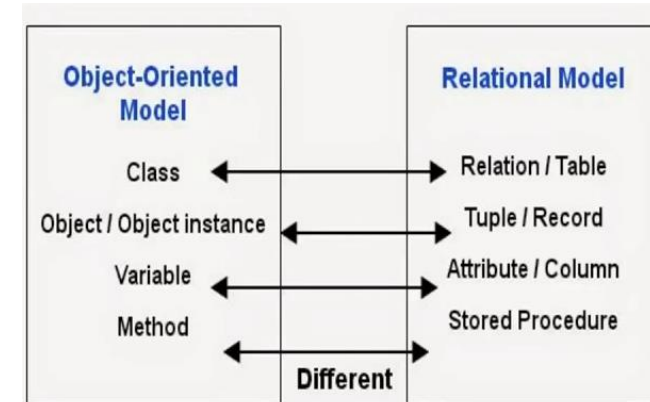
- A strict hierarchical arrangement is not possible when an employee works for two departments, then the tree will become graph.
- In hierarchical model we have to place a data simultaneously in two location in the list. In network we can connect.
 - There are two limitations: Similar to hierarchical databases, network databases must be defined in advance.
 - There is also a limit to the number of connections that can be made between records.



Object-oriented Model

OOM represents an entity as a class.

- A class represents both object attributes as well as the behavior of the entity.
- For example, a book class will have not only the book attributes (ISBN, Title, Price) but also procedure that imitate actions expected (updateprice()) of a book.
- Object-oriented databases have two disadvantages. First, they are more costly to develop. Second, most organizations are reluctant to abandon or convert from those databases that they have already invested money in developing and implementing.

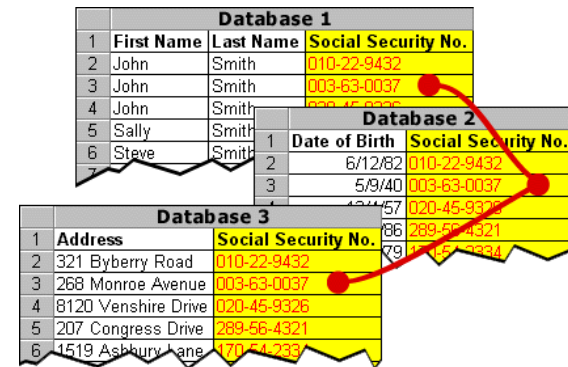


(Object-Oriented database is product of OOP and RDB)

Object-Oriented database

Relational Model

- Hierarchical and network databases require the user to pass down through a hierarchy in order to access needed data.
- Relational databases connect data in different files by using common data elements or a key field.
- Relational databases are more flexible than either the hierarchical or network database structures.



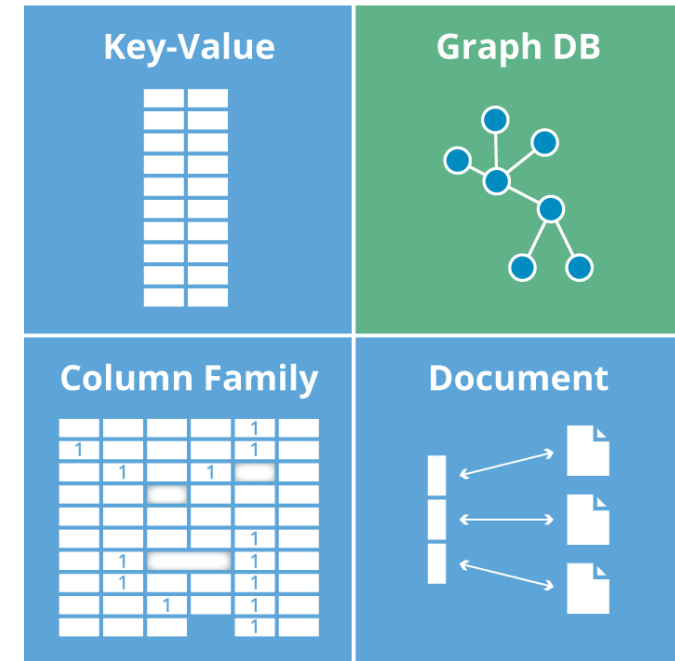
	First Name	Last Name	Social Security No.
1	John	Smith	010-22-9432
2	John	Smith	003-63-0037
3	John	Smith	003-63-0037
4	John	Smith	003-63-0037
5	Sally	Smith	003-63-0037
6	Steve	Smith	003-63-0037
7			

	Date of Birth	Social Security No.
1	6/12/82	010-22-9432
2	5/9/40	003-63-0037
3	12/11/57	020-45-9326
4	08/05/86	289-56-4321
5	07/05/79	170-54-2334

	Address	Social Security No.
1	321 Byberry Road	010-22-9432
2	268 Monroe Avenue	003-63-0037
3	8120 Venshire Drive	020-45-9326
4	207 Congress Drive	289-56-4321
5	1519 Ashbury Lane	170-54-2334
6		

NoSQL Databases

- **Key-Value Store** – It has a Big Hash Table of keys & values {Example- Redis, DynamoDb}
- **Document-based Store**- It stores documents made up of tagged elements. {Example- MongoDB, CouchDB}
- **Column-based Store**- Each storage block contains data from only one column, {Example- Cassandra, HBase}
- **Graph-based**-A network database that uses edges and nodes to represent and store data. {Example- Neo4J}



Data Management for Data Science

Relational Databases

Başar Öztayşı
oztaysib@itu.edu.tr



Problems with Lists: Redundancy

- In a list, each row is intended to stand on its own. As a result, the same information may be entered several times (i.e., **redundancy**)

SALES_ORDERS					
SO_Number	Item_Number	Item_Name	Qty_Ordered	Cust_Code	Cust_Name
1010	2010-0050	Formed Handlebar	2	WHEEL	Wheelaway Cycle Center
	1000-1	20 in. Bicycle	5	WHEEL	Wheelaway Cycle Center
1011	1002-1	24 in. Bicycle	5	ETC	Bikes Et Cetera
	1001-1	26 in. Bicycle	10	ETC	Bikes Et Cetera
1012	1003-1	20 in. Bicycle	5	WHEEL	Wheelaway Cycle Center
	1001-1	26 in. Bicycle	10	WHEEL	Wheelaway Cycle Center
1013	1001-1	26 in. Bicycle	50	IBS	Inter. Bicycle Sales
1014	1003-1	20 in. Bicycle	25	ETC	Bikes Et Cetera
1015	1003-1	20 in. Bicycle	25	WHEEL	Wheelaway Cycle Center
1016	3961-1041	Tire Tube, 26 in.	5	ETC	Bikes Et Cetera
	3965-1050	Spoke Reflector	50	ETC	Bikes Et Cetera
	1003-1	20 in. Bicycle	5	ETC	Bikes Et Cetera
	1000-1	20 in. Bicycle	4	ETC	Bikes Et Cetera

Problems with Lists: Multiple Themes

- In a list, each row may contain information on more than one “theme”. As a result, needed information may appear in the lists only if information on other themes is also present

SALES_ORDERS					
SO_Number	Item_Number	Item_Name	Qty_Ordered	Cust_Code	Cust_Name
1010	2010-0050	Formed Handlebar	2	WHEEL	Wheelaway Cycle Center
	1000-1	20 in. Bicycle	5	WHEEL	Wheelaway Cycle Center
1011	1002-1	24 in. Bicycle	5	ETC	Bikes Et Cetera
	1001-1	26 in. Bicycle	10	ETC	Bikes Et Cetera
1012	1003-1	20 in. Bicycle	5	WHEEL	Wheelaway Cycle Center
	1001-1	26 in. Bicycle	10	WHEEL	Wheelaway Cycle Center
1013	1001-1	26 in. Bicycle	50	IBS	Inter. Bicycle Sales
1014	1003-1	20 in. Bicycle	25	ETC	Bikes Et Cetera
1015	1003-1	20 in. Bicycle	25	WHEEL	Wheelaway Cycle Center
1016	3961-1041	Tire Tube, 26 in.	5	ETC	Bikes Et Cetera
	3965-1050	Spoke Reflector	50	ETC	Bikes Et Cetera
	1003-1	20 in. Bicycle	5	ETC	Bikes Et Cetera
	1000-1	20 in. Bicycle	4	ETC	Bikes Et Cetera

This table includes information on the Sales Order, Items, and Customer

Inter. Bicycle Sales information would not be available without SO #1013

List Modification Issues

- Redundancy and multiple themes create modification problems

Deletion problems

- *If we delete SO #1013 we lose our customer IBS*




Update problems

- *If customer “Bikes Et Cetera” changes its name we have to make sure we change it in every single row.*

Insertion problems

- *If we want to add a new customer, we must either wait until they place an order or (worse) make up a bogus sales order*

List Modification Issues

SALES_ORDERS						
SO_Number	Item_Number	Item_Name	Qty_Ordered	Cust_Code	Cust_Name	
1010	2010-0050	Formed Handleba	2	WHEEL	Wheelaway Cycle Center	
	1000-1	20 in. Bicycle	5	WHEEL	Wheelaway Cycle Center	
1011	1002-1	24 in. Bicycle	5	ETC	Bikes Et Cetera	
	1001-1	26 in. Bicycle	10	ETC	Bikes Et Cetera	
1012	1003-1	20 in. Bicycle	5	WHEEL	Wheelaway Cycle Center	
	1001-1	26 in. Bicycle	10	WHEEL	Wheelaway Cycle Center	
	1013	1001-1	26 in. Bicycle	50	IBS	
	1014	1003-1	20 in. Bicycle	25	ETC	Bikes Et Cetera
	1015	1003-1	20 in. Bicycle	25	WHEEL	Wheelaway Cycle Center
	1016	3961-1041	Tire Tube, 26 in.	5	ETC	Bikes etc.
		3965-1050	Spoke Reflector	50	ETC	Bikes etc.
		1003-1	20 in. Bicycle	5	ETC	Bikes etc.
		1000-1	20 in. Bicycle	4	ETC	Bikes etc.
					New (& rich) Customer	

DELETE
No SO
#1013,
no IBS

UPDATE
Need to
change
Et Cetera
to etc.

INSERT
Blank fields
may cause
problems
later

RDBMS - Tables

SALES_ORDERS			
SO_Number	Item_Number	Item_Name	Qty_Ordered
1010	2010-0050	Formed Handleba	2
	1000-1	20 in. Bicycle	5
1011	1002-1	24 in. Bicycle	5
	1001-1	26 in. Bicycle	10
1012	1003-1	20 in. Bicycle	5
	1001-1	26 in. Bicycle	10
1013	1001-1	26 in. Bicycle	50
1014	1003-1	20 in. Bicycle	25
1015	1003-1	20 in. Bicycle	25
1016	3961-1041	Tire Tube, 26 in.	5
	3965-1050	Spoke Reflector	50
	1003-1	20 in. Bicycle	5
	1000-1	20 in. Bicycle	4

CUSTOMERS	
Cust_Code	Cust_Name
WHEEL	Wheelaway Cycle Center
ETC	Bikes Et Cetera
IBS	Inter. Bicycle Sales

SALES_ORDERS				
SO_Number	Item_Number	Item_Name	Qty_Ordered	Cust_Code
1010	2010-0050	Formed Handleba	2	1
	1000-1	20 in. Bicycle	5	1
1011	1002-1	24 in. Bicycle	5	2
	1001-1	26 in. Bicycle	10	2
1012	1003-1	20 in. Bicycle	5	1
	1001-1	26 in. Bicycle	10	1
1013	1001-1	26 in. Bicycle	50	3
1014	1003-1	20 in. Bicycle	25	2
1015	1003-1	20 in. Bicycle	25	1
1016	3961-1041	Tire Tube, 26 in.	5	2
	3965-1050	Spoke Reflector	50	2
	1003-1	20 in. Bicycle	5	2
	1000-1	20 in. Bicycle	4	2

CUSTOMERS		
Cust_Id	Cust_Code	Cust_Name
1	WHEEL	Wheelaway Cycle Center
2	ETC	Bikes Et Cetera
3	IBS	Inter. Bicycle Sales

MYSQL DATA TYPES

Date data types:

Data type	Description
DATE()	A date. Format: YYYY-MM-DD Note: The supported range is from '1000-01-01' to '9999-12-31'
DATETIME()	*A date and time combination. Format: YYYY-MM-DD HH:MI:SS Note: The supported range is from '1000-01-01 00:00:00' to '9999-12-31 23:59:59'
TIMESTAMP()	*A timestamp. TIMESTAMP values are stored as the number of seconds since the Unix epoch ('1970-01-01 00:00:00' UTC). Format: YYYY-MM-DD HH:MI:SS Note: The supported range is from '1970-01-01 00:00:01' UTC to '2038-01-09 03:14:07' UTC
TIME()	A time. Format: HH:MI:SS Note: The supported range is from '-838:59:59' to '838:59:59'
YEAR()	A year in two-digit or four-digit format. Note: Values allowed in four-digit format: 1901 to 2155. Values allowed in two-digit format: 70 to 69, representing years from 1970 to 2069

MSSQL DATA TYPES

String data types:

Data type	Description	Max size	Storage
char(n)	Fixed width character string	8,000 characters	Defined width
varchar(n)	Variable width character string	8,000 characters	2 bytes + number of chars
varchar(max)	Variable width character string	1,073,741,824 characters	2 bytes + number of chars
text	Variable width character string	2GB of text data	4 bytes + number of chars
nchar	Fixed width Unicode string	4,000 characters	Defined width x 2
nvarchar	Variable width Unicode string	4,000 characters	
nvarchar(max)	Variable width Unicode string	536,870,912 characters	
ntext	Variable width Unicode string	2GB of text data	
binary(n)	Fixed width binary string	8,000 bytes	
varbinary	Variable width binary string	8,000 bytes	
varbinary(max)	Variable width binary string	2GB	
image	Variable width binary string	2GB	

MSSQL DATA TYPES

Number data types:

Data type	Description	Storage
bit	Integer that can be 0, 1, or NULL	
tinyint	Allows whole numbers from 0 to 255	1 byte
smallint	Allows whole numbers between -32,768 and 32,767	2 bytes
int	Allows whole numbers between -2,147,483,648 and 2,147,483,647	4 bytes
bigint	Allows whole numbers between -9,223,372,036,854,775,808 and 9,223,372,036,854,775,807	8 bytes
decimal(p,s)	Fixed precision and scale numbers. Allows numbers from $-10^{38} + 1$ to $10^{38} - 1$. The p parameter indicates the maximum total number of digits that can be stored (both to the left and to the right of the decimal point). p must be a value from 1 to 38. Default is 18. The s parameter indicates the maximum number of digits stored to the right of the decimal point. s must be a value from 0 to p. Default value is 0	5-17 bytes
numeric(p,s)	Fixed precision and scale numbers. Allows numbers from $-10^{38} + 1$ to $10^{38} - 1$. The p parameter indicates the maximum total number of digits that can be stored (both to the left and to the right of the decimal point). p must be a value from 1 to 38. Default is 18. The s parameter indicates the maximum number of digits stored to the right of the decimal point. s must be a value from 0 to p. Default value is 0	5-17 bytes
smallmoney	Monetary data from -214,748.3648 to 214,748.3647	4 bytes
money	Monetary data from -922,337,203,685,477.5808 to 922,337,203,685,477.5807	8 bytes
float(n)	Floating precision number data from $-1.79E + 308$ to $1.79E + 308$. The n parameter indicates whether the field should hold 4 or 8 bytes. float(24) holds a 4-byte field and float(53) holds an 8-byte field. Default value of n is 53.	4 or 8 bytes
real	Floating precision number data from $-3.40E + 38$ to $3.40E + 38$	4 bytes

Data Management for Data Science

Introduction to SQL

Başar Öztayşı
oztaysib@itu.edu.tr



Structured Query Language

- SQL is an ANSI (American National Standards Institute) standard computer language for accessing and manipulating database systems.
- SQL works with database programs like MS Access,
- DB2,
- Informix,
- MS SQL Server,
- Oracle,
- Sybase, etc.



- There are different versions of the SQL language, but they must support the same major keywords in a similar manner (such as SELECT, UPDATE, DELETE, INSERT, WHERE, and others) with SQL standards
- Most of the SQL database programs also have their own extensions in addition to the SQL standard!



- SQL can be divided into two parts:
 - The Data Manipulation Language (DML)
 - The Data Definition Language (DDL).

- SQL (Structured Query Language) is a syntax for executing queries. But the SQL language also includes a syntax to update, insert, and delete records.
 - SELECT - extracts data from a database table
 - UPDATE - updates data in a database table
 - DELETE - deletes data from a database table
 - INSERT INTO - inserts new data into a database table



- The Data Definition Language (DDL) part of SQL permits database tables to be created or deleted. We can also define indexes (keys), specify links between tables, and impose constraints between database tables.
- The most important DDL statements in SQL are:
 - CREATE TABLE - creates a new database table
 - ALTER TABLE - alters (changes) a database table
 - DROP TABLE - deletes a database table
 - CREATE INDEX - creates an index (search key)
 - DROP INDEX - deletes an index

SQL The SELECT Statement

- The SELECT statement is used to select data from a table. The tabular result is stored in a result table (called the result-set).

Syntax

```
SELECT column_name(s)  
FROM table_name
```



Filtering Columns

LastName	FirstName	Address	City
Kiraz	Meltem	Paşa Sokak 1/23	İzmit
Bohçacı	Berke	Çeşme Sokak 4/1	İzmit
Sungur	Ayşegül	Tepe Sokak 8/23	Balıkesir

To select the columns named "LastName" and "FirstName", use a SELECT statement like this:

```
SELECT LastName, FirstName FROM Person
```

LastName	FirstName
Kiraz	Meltem
Bohçacı	Berke
Sungur	Ayşegül

Select All Columns

To select all columns from the "Person" table, use a * symbol instead of column names, like this:

```
SELECT * FROM Person
```

LastName	FirstName	Address	City
Kiraz	Meltem	Paşa Sokak 1/23	İzmit
Bohçacı	Berke	Çeşme Sokak 4/1	İzmit
Sungur	Ayşegül	Tepe Sokak 8/23	Balıkesir

Select Distinct Columns

The DISTINCT keyword is used to return only distinct (different) values.

The SELECT statement returns information from table columns. But what if we only want to select distinct elements?

With SQL, all we need to do is to add a DISTINCT keyword to the SELECT statement:

Syntax

SELECT DISTINCT **column_name(s)**

FROM **table_name**



Select Distinct Columns

To select ALL values from the column named "Company" we use a SELECT statement like this:

```
SELECT Company FROM Orders
```

Orders

Company	OrderNumber
SONY	3412
ACME	5464
VESTEL	5876
ACME	3425



Company
SONY
ACME
VESTEL
ACME

Select Distinct Columns

Note that "W3Schools" is listed twice in the result-set.

To select only DIFFERENT values from the column named "Company" we use a SELECT DISTINCT statement like this:

```
SELECT DISTINCT Company FROM Orders
```

Orders

Company	OrderNumber
SONY	3412
ACME	5464
VESTEL	5876
ACME	3425



Company
SONY
ACME
VESTEL

The WHERE clause is used to specify a selection criterion.

The WHERE Clause

To conditionally select data from a table, a WHERE clause can be added to the SELECT statement.

Syntax

```
SELECT column FROM table  
WHERE column operator value
```



With the WHERE clause, the following operators can be used:

Operator	Description
=	Equal
<>	Not equal
>	Greater than
<	Less than
>=	Greater than or equal
<=	Less than or equal
BETWEEN	Between an inclusive range
LIKE	Search for a pattern
IN	If you know the exact value you want to return for at least one of the columns

Using the WHERE Clause

To select only the persons living in the city "İzmit", we add a WHERE clause to the SELECT statement:

```
SELECT * FROM Person  
WHERE City='İzmit'
```

LastName	FirstName	Address	City
Kiraz	Meltem	Paşa Sokak 1/23	İzmit
Bohçacı	Berke	Çeşme Sokak 4/1	İzmit
Sungur	Ayşegül	Tepe Sokak 8/23	Balıkesir
Bohçacı	Nilgün	Kahya Sokak 3/65	İzmit

LastName	FirstName	Address	City
Kiraz	Meltem	Paşa Sokak 1/23	İzmit
Bohçacı	Berke	Çeşme Sokak 4/1	İzmit
Bohçacı	Nilgün	Kahya Sokak 3/65	İzmit

Using Quotes

SQL uses single quotes around text values Numeric values should not be enclosed in quotes.

For Text Values

This is correct:

```
SELECT * FROM Person WHERE FirstName='Ayşe'
```

This is wrong:

```
SELECT * FROM Person WHERE FirstName=Ayşe
```

For Numeric Values

This is correct:

```
SELECT * FROM Person WHERE Year=1985
```

This is wrong:

```
SELECT * FROM Person WHERE Year='1985'
```

The LIKE Condition

The LIKE condition is used to specify a search for a pattern in a column.

Syntax

```
SELECT column FROM table  
WHERE column LIKE pattern
```

A "%" sign can be used to define wildcards (missing letters in the pattern) both before and after the pattern.



Using LIKE

The following SQL statement will return persons with first names that start with an 'O':

```
SELECT * FROM Person  
WHERE FirstName LIKE 'O%'
```

The following SQL statement will return persons with first names that end with an 'a':

```
SELECT * FROM Person  
WHERE FirstName LIKE '%a'
```



Using LIKE ₂

The following SQL statement will return persons with first names that contain the pattern 'la':

```
SELECT * FROM Person WHERE FirstName LIKE '%la%'
```

```
SELECT * FROM Person WHERE (FirstName LIKE '%la%' )  
AND (FirstName LIKE '%O' )
```

```
SELECT * FROM Person WHERE ((FirstName LIKE '%la%'  
) AND (FirstName LIKE '%O' ))
```

```
SELECT * FROM Person WHERE ((FirstName LIKE '%la%'  
) AND (City = 'İzmit' )) OR (Lastname = 'Özdemir')
```

SQL The INSERT INTO Statement

The INSERT INTO Statement

The INSERT INTO statement is used to insert new rows into a table.

Syntax

```
INSERT INTO table_name  
VALUES (value1, value2,....)
```

You can also specify the columns for which you want to insert data:

```
INSERT INTO table_name (column1, column2,...)  
VALUES (value1, value2,....)
```

Insert a New Row

LastName	FirstName	Address	City
Kiraz	Meltem	Paşa Sokak 1/23	İzmit

INSERT INTO Person

VALUES ('Arslan', 'Hüseyin', 'Karga Sok 3/23', 'Balıkesir')

LastName	FirstName	Address	City
Kiraz	Meltem	Paşa Sokak 1/23	İzmit
Arslan	Hüseyin	Karga Sok 3/23	Balıkesir

Insert Data in Specified Columns

LastName	FirstName	Address	City
Kiraz	Meltem	Paşa Sokak 1/23	İzmit
Arslan	Hüseyin	Karga Sok 3/23	Balıkesir

And This SQL statement:

```
INSERT INTO Person (LastName, Address)  
VALUES ('Kibar', 'Terzi Sok 1/3')
```

LastName	FirstName	Address	City
Kiraz	Meltem	Paşa Sokak 1/23	İzmit
Arslan	Hüseyin	Karga Sok 3/23	Balıkesir
Kibar		Terzi Sok 1/3	

SQL The UPDATE Statement

The Update Statement

The UPDATE statement is used to modify the data in a table.

Syntax

UPDATE **table_name**

SET **column_name = new_value**

WHERE **column_name = some_value**



Update one Column in a Row

LastName	FirstName	Address	City
Kiraz	Meltem	Paşa Sokak 1/23	İzmit
Arslan	Hüseyin	Karga Sok 3/23	Balıkesir
Kibar		Terzi Sok 1/3	

We want to add a first name to the person with a last name of "Kibar":

```
UPDATE Person SET FirstName = 'Nehar'  
WHERE LastName = 'Kibar'
```

LastName	FirstName	Address	City
Kiraz	Meltem	Paşa Sokak 1/23	İzmit
Arslan	Hüseyin	Karga Sok 3/23	Balıkesir
Kibar	Nehar	Terzi Sok 1/3	

Update several Columns in a Row

LastName	FirstName	Address	City
Kiraz	Meltem	Paşa Sokak 1/23	İzmit
Arslan	Hüseyin	Karga Sok 3/23	Balıkesir
Kibar	Nehar	Terzi Sok 1/3	

We want to change the address and add the name of the city:

```
UPDATE Person
```

```
SET Address = 'Keresteci Sok 1/7', City = 'İzmit'
```

```
WHERE LastName = 'Kibar'
```

LastName	FirstName	Address	City
Kiraz	Meltem	Paşa Sokak 1/23	İzmit
Arslan	Hüseyin	Karga Sok 3/23	Balıkesir
Kibar	Nehar	Keresteci Sok 1/7	İzmit

SQL The Delete Statement

The Delete Statement

The DELETE statement is used to delete rows in a table.

Syntax

```
DELETE FROM table_name  
WHERE column_name = some_value
```



Delete a Row

LastName	FirstName	Address	City
Kiraz	Meltem	Paşa Sokak 1/23	İzmit
Arslan	Hüseyin	Karga Sok 3/23	Balıkesir
Kibar	Nehar	Terzi Sok 1/3	İzmit

"Nehar Kibar" is going to be deleted:

DELETE FROM Person WHERE LastName = 'Kibar'

LastName	FirstName	Address	City
Kiraz	Meltem	Paşa Sokak 1/23	İzmit
Arslan	Hüseyin	Karga Sok 3/23	Balıkesir

LastName	FirstName	Address	City
Kiraz	Meltem	Paşa Sokak 1/23	İzmit
Arslan	Hüseyin	Karga Sok 3/23	Balıkesir
Kibar	Nehar	Terzi Sok 1/3	İzmit

LastName	FirstName	Address	City
Kiraz	Meltem	Paşa Sokak 1/23	İzmit
Arslan	Hüseyin	Karga Sok 3/23	Balıkesir
Kibar	Nehar	Terzi Sok 1/3	

SQL

Advanced Structured Query Language

Top N

- The SELECT TOP clause is used to specify the number of records to return.
- The SELECT TOP clause is useful on large tables with thousands of records. Returning a large number of records can impact on performance.

SQL Server / MS Access Syntax:

```
SELECT TOP number | percent column_name(s)  
FROM table_name  
WHERE condition;
```

MySQL Syntax:

```
SELECT column_name(s)  
FROM table_name  
WHERE condition  
LIMIT number;
```



The SQL MIN() and MAX() Functions

The MIN() function returns the smallest value of the selected column.

The MAX() function returns the largest value of the selected column.

MIN() Syntax

```
SELECT MIN(column_name)
FROM table_name
WHERE condition;
```

MAX() Syntax

```
SELECT MAX(column_name)
FROM table_name
WHERE condition;
```

The SQL COUNT(), AVG() and SUM() Functions

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

The AVG() function returns the average value of a numeric column.

The SUM() function returns the total sum of a numeric column.

COUNT() Syntax

```
SELECT COUNT(column_name)
FROM table_name
WHERE condition;
```

AVG() Syntax

```
SELECT AVG(column_name)
FROM table_name
WHERE condition;
```

SUM() Syntax

```
SELECT SUM(column_name)
FROM table_name
WHERE condition;
```

LIKE Syntax

```
SELECT column1, column2, ...  
FROM table_name  
WHERE columnN LIKE pattern;
```


The SQL IN Operator

The IN operator allows you to specify multiple values in a WHERE clause.

The IN operator is a shorthand for multiple OR conditions.

IN Syntax

```
SELECT column_name(s)
FROM table_name
WHERE column_name IN (value1, value2, ...);
```

or:

```
SELECT column_name(s)
FROM table_name
WHERE column_name IN (SELECT STATEMENT);
```

The SQL BETWEEN Operator

The BETWEEN operator selects values within a given range. The values can be numbers, text, or dates.

The BETWEEN operator is inclusive: begin and end values are included.

BETWEEN Syntax

```
SELECT column_name(s)
FROM table_name
WHERE column_name BETWEEN value1 AND value2;
```

SQL Aliases

SQL aliases are used to give a table, or a column in a table, a temporary name.

Aliases are often used to make column names more readable.

An alias only exists for the duration of the query.

Alias Column Syntax

```
SELECT column_name AS alias_name
FROM table_name;
```

Alias Table Syntax

```
SELECT column_name(s)
FROM table_name AS alias_name;
```

The SQL GROUP BY Statement

The GROUP BY statement is often used with aggregate functions (COUNT, MAX, MIN, SUM, AVG) to group the result-set by one or more columns.

GROUP BY Syntax

```
SELECT column_name(s)
FROM table_name
WHERE condition
GROUP BY column_name(s)
ORDER BY column_name(s);
```

The SQL HAVING Clause

The HAVING clause was added to SQL because the WHERE keyword could not be used with aggregate functions.

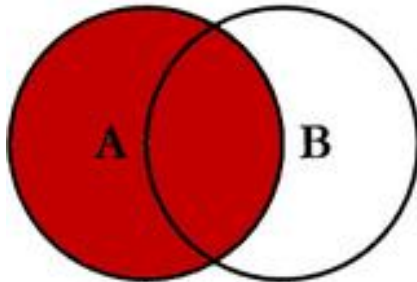
HAVING Syntax

```
SELECT column_name(s)
FROM table_name
WHERE condition
GROUP BY column_name(s)
HAVING condition
ORDER BY column_name(s);
```

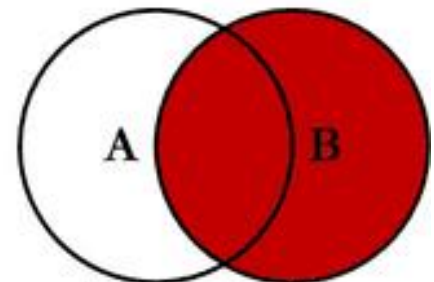
SQL

Joining Tables

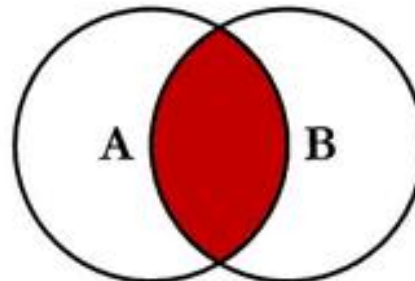
SQL JOINS



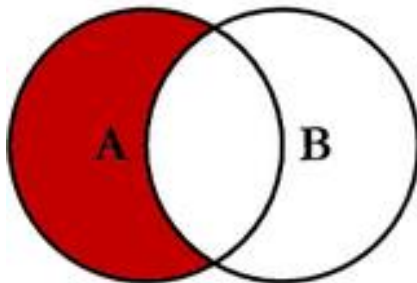
```
SELECT <select_list>
FROM TableA A
LEFT JOIN TableB B
ON A.Key = B.Key
```



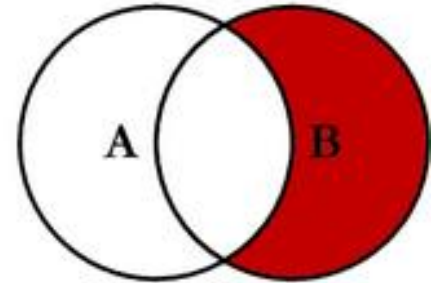
```
SELECT <select_list>
FROM TableA A
RIGHT JOIN TableB B
ON A.Key = B.Key
```



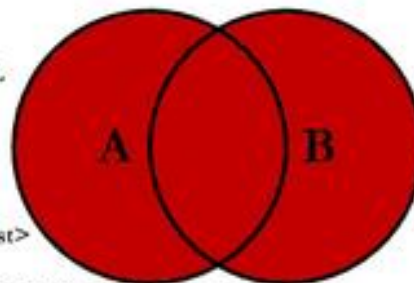
```
SELECT <select_list>
FROM TableA A
INNER JOIN TableB B
ON A.Key = B.Key
```



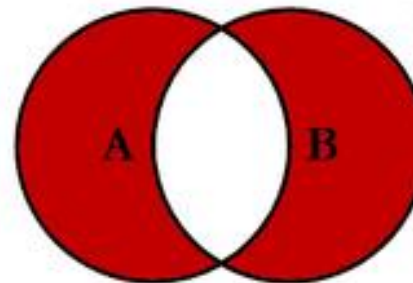
```
SELECT <select_list>
FROM TableA A
LEFT JOIN TableB B
ON A.Key = B.Key
WHERE B.Key IS NULL
```



```
SELECT <select_list>
FROM TableA A
RIGHT JOIN TableB B
ON A.Key = B.Key
WHERE A.Key IS NULL
```



```
SELECT <select_list>
FROM TableA A
FULL OUTER JOIN TableB B
ON A.Key = B.Key
```



```
SELECT <select_list>
FROM TableA A
FULL OUTER JOIN TableB B
ON A.Key = B.Key
WHERE A.Key IS NULL
OR B.Key IS NULL
```

- EQUI JOIN
 - EQUI JOIN is a simple SQL join.
 - Uses the equal sign(=) as the comparison operator for the condition
- NON EQUI JOIN
 - NON EQUI JOIN uses comparison operator other than the equal sign.
 - The operators uses like >, <, >=, <= with the condition.

Types of SQL EQUI JOIN

- INNER JOIN
 - Returns only matched rows from the participating tables.
 - Match happened only at the key record of participating tables.
- OUTER JOIN
 - Returns all rows from one table and
 - Matching rows from the secondary table and
 - Comparison columns should be equal in both the tables.

List of SQL JOINS

- INNER JOIN
- LEFT JOIN OR LEFT OUTER JOIN
- RIGHT JOIN OR RIGHT OUTER JOIN
- FULL OUTER JOIN
- NATURAL JOIN
- CROSS JOIN
- SELF JOIN



INNER JOIN

The INNER JOIN selects all rows from both participating tables as long as there is a match between the columns.

An SQL INNER JOIN is same as JOIN clause, combining rows from two or more tables.

Example: INNER JOIN

```
SELECT * FROM table_A  
INNER JOIN table_B  
ON table_A.A=table_B.A;
```



LEFT JOIN or LEFT OUTER JOIN

The SQL LEFT JOIN, joins two tables and fetches rows based on a condition, which are matching in both the tables.

The unmatched rows will also be available from the table before the JOIN clause.

Example: LEFT JOIN or LEFT OUTER JOIN

```
SELECT * FROM table_A  
LEFT JOIN table_B  
ON table_A.A=table_B.A;
```



RIGHT JOIN or RIGHT OUTER JOIN

The SQL RIGHT JOIN, joins two tables and fetches rows based on a condition, which are matching in both the tables.

The unmatched rows will also be available from the table written after the JOIN clause.

Example : RIGHT JOIN or RIGHT OUTER JOIN

```
SELECT * FROM table_A  
RIGHT JOIN table_B  
ON table_A.A=table_B.A;
```

UNION

- You can use UNION if you want to select rows one after the other from several tables, or several sets of rows from a single table all as a single result set.
- Syntax

SELECT

UNION [ALL | DISTINCT] SELECT

UNION [ALL | DISTINCT] SELECT



UNION

- MySQL uses the names of columns in the first SELECT statement as the labels for the output.

```
SELECT fname, lname, addr FROM prospect  
UNION  
SELECT first_name, last_name, adress FROM  
customer  
UNION  
SELECT company, ' ', street FROM vendor;
```