

Introduction to Operating Systems and SQL for Data Science

Lecture 9 – Structured Query Language (SQL)

Previous lecture

- Relational algebra
- Set operators
- Special Relational operators
- Aggregate functions & grouping

SQL - Introduction

- SQL – Structured Query language
(Original name: SEQUEL – Structured English Query Language)
- Invented by IBM in a research system – System R.
- Nonprocedural language; the operations are more like “What we want to get” and **not** “how to do it”.
- Allows end users to query and update data on the DMBS.
- The DBMS (using optimizer) translate the SQL operators to the relevant procedures efficiently.

SQL is a declarative language

```
SELECT FirstName, LastName  
FROM Users  
WHERE city='Tel-aviv'  
ORDER BY Age
```

- In Python/Java (procedural language) it looks much different.

DDL commands

Data Definition Language (DDL)

DDL – Data Definition Language

- Operations to define the database scheme.
- Most commands are at a logical/central schema level (defining tables, fields, and constraints).
- Some of the commands are at the level of a physical schema (indexes; access permissions).
- Some of the commands are at the external schema level (views)

SQL Schema

- A Schema in SQL is a collection of database objects associated with a database.
- Schema always belong to a single database whereas a database can have single or multiple schemas.

How to create a Schema?

CREATE SCHEMA [schema_name]

AUTHORIZATION [owner_name]

SQL table

- The **create table statement** is used to create a table for the database you are using.
- This table can have n rows and m columns based on the requirement (determine in the design phase).

How to create a table?

```
CREATE TABLE tablename (  
  column1 data type [not null] [UNIQUE][DEAFULT <Value>],  
  column2 data type,  
  column3 data type,  
  column4 data type,  
  ....  
  columnN data type)  
[PRIMARY KEY (column_i,..., column_j)]  
[FOREIGN KEY (column_i,..., column_j) REFERENCES table_name (col_i,...,col_j)  
  ON DELETE CASCADE/SET NULL/NO ACTION/SET DEAFULT  
  ON UPDATE CASCADE/SET NULL/NO ACTION/SET DEAFULT]  
...  
[FOREIGN KEY ...]
```

[] – marked as optional

How to create a table? - continue

- The order of the columns is determined by the order in the CREATE command.
- The order of the records in the table is not defined (in the CREATE command)

How to create a table? - example

Query:

```
CREATE TABLE students (  
  studentID int,  
  studentName varchar(255),  
  parentName varchar(255),  
  address varchar(255),  
  phonenumber int  
);
```

Output:

studentID	studentName	parentName	address	phoneNumber
-----------	-------------	------------	---------	-------------

Table data types

- CHAR (n) – string in length n.
- DEC(n, m) – a decimal number with n digits, which m digits are after the decimal separator.
- INTEGER – hold small whole numbers, INTEGER values have 32 bits; represent whole numbers from $-2^{31}-1$ through $2^{31}-1$.
- SMALLINT - hold small whole numbers, SMALLINT values have 16 bits; represent whole numbers from $-2^{15}-1$ through $2^{15}-1$.
- DOUBLE - hold non-whole numbers
- FLOAT – hold small non-whole numbers.
- DATE – date in agreed format (e.g., YYYY/MM/DD)

Create table – book example

```
CREATE TABLE EMPLOYEE
( Fname          VARCHAR(15)          NOT NULL,
  Minit          CHAR,
  Lname          VARCHAR(15)          NOT NULL,
  Ssn            CHAR(9)              NOT NULL,
  Bdate          DATE,
  Address        VARCHAR(30),
  Sex            CHAR,
  Salary         DECIMAL(10,2),
  Super_ssn      CHAR(9),
  Dno            INT                  NOT NULL,
  PRIMARY KEY (Ssn),
  FOREIGN KEY (Super_ssn) REFERENCES EMPLOYEE(Ssn),
  FOREIGN KEY (Dno) REFERENCES DEPARTMENT(Dnumber) );

CREATE TABLE DEPARTMENT
( Dname          VARCHAR(15)          NOT NULL,
  Dnumber        INT                  NOT NULL,
  Mgr_ssn        CHAR(9)              NOT NULL,
  Mgr_start_date DATE,
  PRIMARY KEY (Dnumber),
  UNIQUE (Dname),
  FOREIGN KEY (Mgr_ssn) REFERENCES EMPLOYEE(Ssn) );

CREATE TABLE DEPT_LOCATIONS
( Dnumber        INT                  NOT NULL,
  Dlocation      VARCHAR(15)          NOT NULL,
  PRIMARY KEY (Dnumber, Dlocation),
  FOREIGN KEY (Dnumber) REFERENCES DEPARTMENT(Dnumber) );
```

Create table – book example

```
CREATE TABLE PROJECT
( Pname          VARCHAR(15)          NOT NULL,
  Pnumber        INT                  NOT NULL,
  Plocation      VARCHAR(15),
  Dnum           INT                  NOT NULL,
  PRIMARY KEY (Pnumber),
  UNIQUE (Pname),
  FOREIGN KEY (Dnum) REFERENCES DEPARTMENT(Dnumber) );

CREATE TABLE WORKS_ON
( Essn           CHAR(9)              NOT NULL,
  Pno            INT                  NOT NULL,
  Hours          DECIMAL(3,1)         NOT NULL,
  PRIMARY KEY (Essn, Pno),
  FOREIGN KEY (Essn) REFERENCES EMPLOYEE(Ssn),
  FOREIGN KEY (Pno) REFERENCES PROJECT(Pnumber) );

CREATE TABLE DEPENDENT
( Essn           CHAR(9)              NOT NULL,
  Dependent_name VARCHAR(15)          NOT NULL,
  Sex            CHAR,
  Bdate          DATE,
  Relationship    VARCHAR(8),
  PRIMARY KEY (Essn, Dependent_name),
  FOREIGN KEY (Essn) REFERENCES EMPLOYEE(Ssn) );
```

Create table – book example

EMPLOYEE	FNAME	MINIT	LNAME	SSN	BDATE	ADDRESS	SEX	SALARY	SUPERSSN	DNAME
	John	B	Smith	123456789	1965-01-09	731 Fondren, Houston, TX	M	30000	333445555	5
	Franklin	T	Wong	333445555	1955-12-08	638 Voss, Houston, TX	M	40000	888665555	5
	Alicia	J	Zelaya	999887777	1968-07-19	3321 Castle, Spring, TX	F	25000	987654321	4
	Jennifer	S	Wallace	987654321	1941-06-20	291 Berry, Bellaire, TX	F	43000	888665555	4
	Ramesh	K	Narayan	666884444	1962-09-15	975 Fire Oak, Humble, TX	M	38000	333445555	5
	Joyce	A	English	453453453	1972-07-31	5631 Rice, Houston, TX	F	25000	333445555	5
	Ahmad	V	Jabbar	987987987	1969-03-29	980 Dallas, Houston, TX	M	25000	987654321	4
	James	E	Borg	888665555	1937-11-10	450 Stone, Houston, TX	M	55000	null	1

DEPT_LOCATIONS					DNUMBER	DLOCATION
DEPARTMENT	DNAME	DNUMBER	MGRSSN	MGRSTARTDATE	1	Houston
	Research	5	333445555	1988-05-22	4	Stafford
	Administration	4	987654321	1995-01-01	5	Bellaire
	Headquarters	1	888665555	1981-06-19	5	Sugarland
					5	Houston

WORKS_ON	ESSN	PNO	HOURS
	123456789	1	32.5
	123456789	2	7.5
	666884444	3	40.0
	453453453	1	20.0
	453453453	2	20.0
	333445555	2	10.0
	333445555	3	10.0
	333445555	10	10.0
	333445555	20	10.0
	999887777	30	30.0
	999887777	10	10.0
	987987987	10	35.0
	987987987	30	5.0
	987654321	30	20.0
	987654321	20	15.0
	888665555	20	null

PROJECT	PNAME	PNUMBER	PLOCATION	DNUM
	ProductX	1	Bellaire	5
	ProductY	2	Sugarland	5
	ProductZ	3	Houston	5
	Computerization	10	Stafford	4
	Reorganization	20	Houston	1
	Newbenefits	30	Stafford	4

DEPENDENT	ESSN	DEPENDENT_NAME	SEX	BDATE	RELATIONSHIP
	333445555	Alice	F	1986-04-05	DAUGHTER
	333445555	Theodore	M	1983-10-25	SON
	333445555	Joy	F	1958-05-03	SPOUSE
	987654321	Abner	M	1942-02-28	SPOUSE
	123456789	Michael	M	1988-01-04	SON
	123456789	Alice	F	1988-12-30	DAUGHTER
	123456789	Elizabeth	F	1967-05-05	SPOUSE

How to change(/alter) a table?

Alter Table <table_name>

- Add attributes:

ADD <attribute name> <data type>

- Drop attributes: (cascade – drop all reference, restricted – the dropping action will occur only if there are not reference)

DROP <attribute_name> {cascade/restricted}

- Add or change an attribute default value

ALTER <attribute_name> {drop default/set default}

How to change(/alter) a table? - example

- If we want to remove the column CourseID from Grades table:

```
ALTER TABLE Grades DROP CourseID
```

- If we want to add phone column to the Student table:

```
ALTER TABLE Students ADD Phone CHAR (7) DEAFULT  
'UNLISTED'
```

How to delete a table?

- If we want to delete a table:

`DROP TABLE <table_name> {cascade/restricted}`

Drop the table and
every reference

Drop the table if there
is no reference to it

- For example, delete Grades table:

`DROP TABLE Grades`

DML commands

Data Manipulation Language (DML)

- Data Manipulation Language or DML represents a collection of programming languages explicitly used to make changes in the database, such as CRUD:

- ☐ Create

- ☐ Read

- ☐ Update

- ☐ Delete

DML commands

In SQL we use the following commands to support CRUD:

INSERT - insert data into a table (create)

SELECT - retrieve data from a database (read)

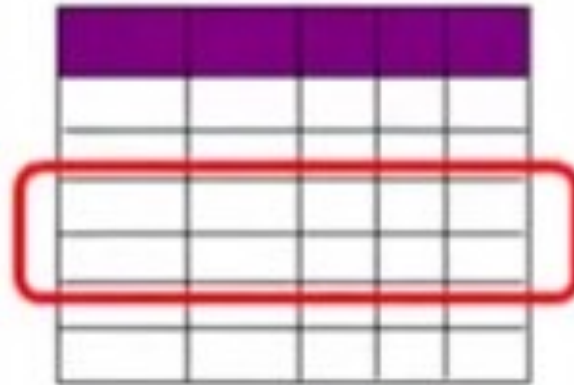
UPDATE - updates existing data within a table

DELETE - Delete all records from a database table

SELECT command

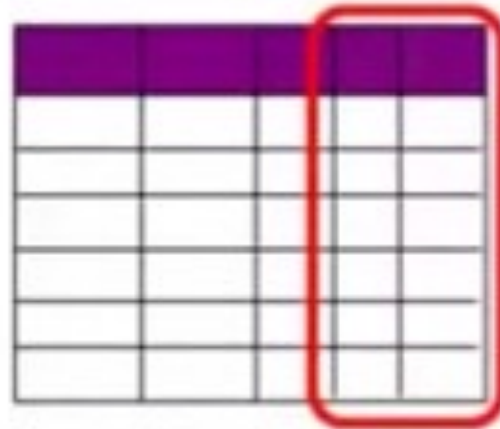
SELECT(projection)	Columns names
FROM	Table names
WHERE (selection)	predicate
GROUP BY	List of column name
HAVING	Predicate
ORDER BY	List of column names

Selection & Projection



A 6x5 grid representing a table. The top row is shaded purple. A red rounded rectangle highlights the second, third, and fourth rows, representing the selection of specific rows from the table.

select



A 6x5 grid representing a table. The top row is shaded purple. A red rounded rectangle highlights the last two columns (columns 4 and 5) across all rows, representing the projection of specific columns from the table.

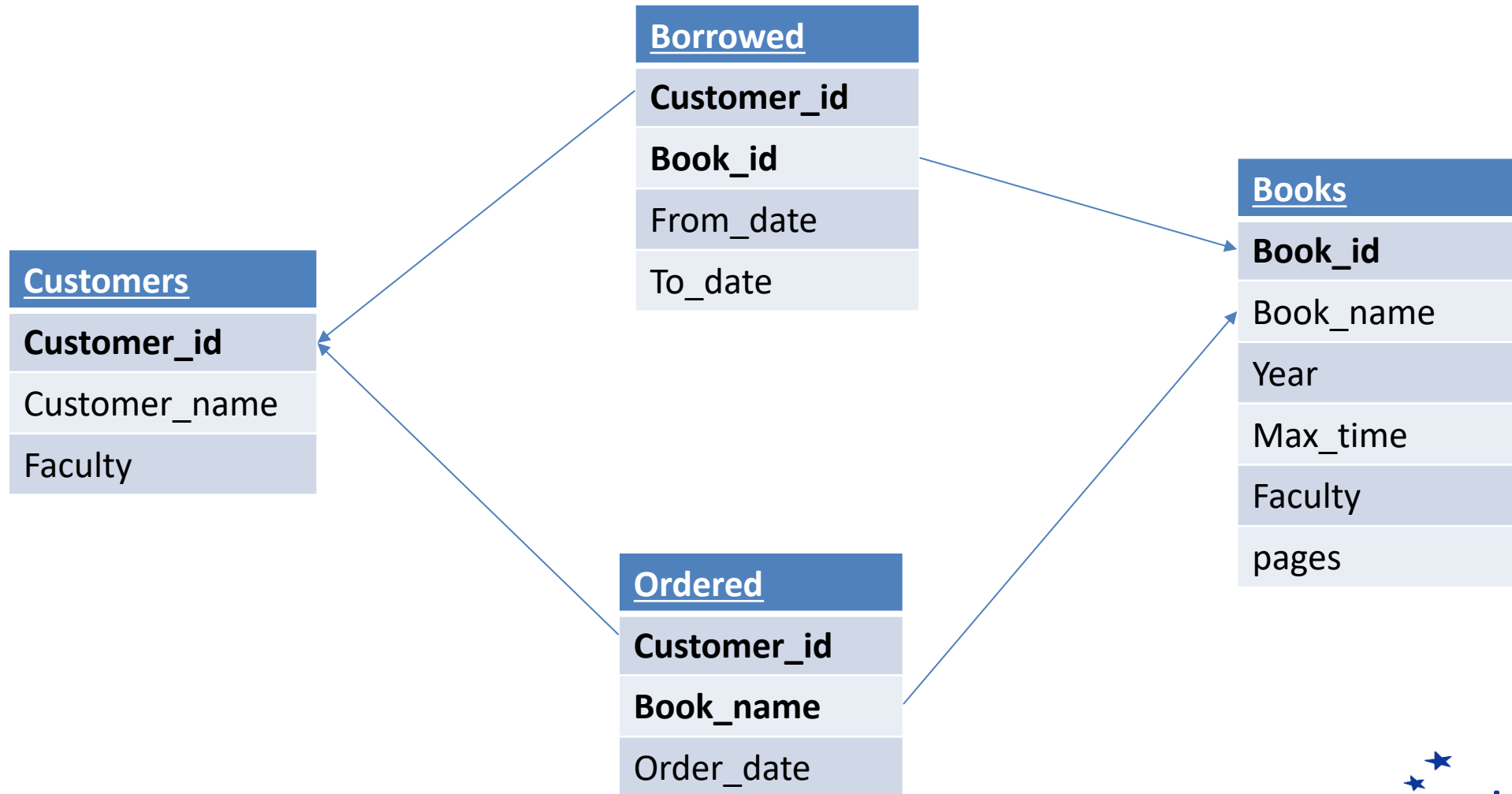
project

SELECT command - continue

- SELECT – the name of the columns we need to retrieve (include * - which means all columns)
- FROM – the names of the tables to retrieve
- WHERE – the predicate on the records
 - Relational operators - <, >, =, <>, =<, =>.
 - Logical operators – NOT, OR, AND

Library example

Library database example



Customer table

Customers (**customer_id**, customer_name, faculty):

- **Customer_id** – a unique identifier for each customer (such as social security number SSN or passport number).
- customer_name – the customer's full name.
- Faculty – the faculty's name.

Customer table - continue

Customer_id	Customer_name	Faculty
12345	Moshe Cohen	CS
23456	Avi Barak	EE
34567	Avi Barak	MED
45678	Lior Edri	EE
56789	Moshe Cohen	EE
67890	Moshe Cohen	EE

Books table

Books(**book_id**, book_name, year, max_time, faculty, pages)

- **Book_id** – a unique identifier for each copy
- Book_name – the book name
- Year – Year of publication
- Max_time – the maximum borrowing time
- Faculty – faculty name
- Pages – the number of pages

Books table - continue

<u>Book_Id</u>	Book_Name	Year	Max_Time	Faculty	Pages
1111	Database Systems	1998	7	CS	348
1112	Database Systems	1998	14	CS	348
1113	Database Systems	2001	7	CS	424
2222	Database And Knowledge	1998	1	CS	390
2223	Database And Knowledge	1998	7	EE	390
3333	Electronic Circuits	1998	21	EE	180
4444	Genes 7	1985	7	MED	580
5555	Anatomy	1988	7	MED	450

Ordered table

Ordered(**customer_id**, **book_name**, order_date)

- **Customer_id** – a unique identifier for each customer (such as social security number SSN or passport number).
- **Book_name** – the book name
- Order_date – the date of order the book copy

Ordered table - continue

Customer_Id	Book_Name	Order_Date
12345	Database Systems	14-Oct-2002
45678	Anatomy	24-Oct-2002
12345	Database And Knowledge	30-Oct-2002
45678	Electronic Circuits	12-Oct-2002

Borrowed table

Borrowed(**book_id**, **customer_id**, from_date, to_date)

- **Book_id** – a unique identifier for each copy.
- **Customer_id** – a unique identifier for each customer (such as social security number SSN or passport number).
- From_date – date of borrowing the book
- To_date – date of return of the book

Borrowed table - continue

Book_id	Customer_Id	From_date	To_Date
5555	56789	13-oct-2022	

Select - example

Books:

<u>Book_Id</u>	Book_Name	Year	Max_Time	Faculty	Pages
1111	Database Systems	1998	7	CS	348
1112	Database Systems	1998	14	CS	348
1113	Database Systems	2001	7	CS	424
2222	Database And Knowledge	1998	1	CS	390
2223	Database And Knowledge	1998	7	EE	390
3333	Electronic Circuits	1998	21	EE	180
4444	Genes 7	1985	7	MED	580
5555	Anatomy	1988	7	MED	450

Select - example

SELECT Book_Name, Pages FROM Books:

Book_Name	Pages
Database Systems	348
Database Systems	348
Database Systems	424
Database And Knowledge	390
Database And Knowledge	390
Electronic Circuits	180
Genes 7	580
Anatomy	450

Select all columns

In order to project all the columns, instead of writing the names of all the columns we can use the character "*" (asterisk).

For example, to project all the columns of Book table:

```
SELECT * FROM Books
```

Select all columns - example

SELECT * FROM Books

<u>Book_Id</u>	Book_Name	Year	Max_Time	Faculty	Pages
1111	Database Systems	1998	7	CS	348
1112	Database Systems	1998	14	CS	348
1113	Database Systems	2001	7	CS	424
2222	Database And Knowledge	1998	1	CS	390
2223	Database And Knowledge	1998	7	EE	390
3333	Electronic Circuits	1998	21	EE	180
4444	Genes 7	1985	7	MED	580
5555	Anatomy	1988	7	MED	450

Arithmetic expressions

- Arithmetic expressions can be retrieved using table fields.
- Example: Retrieving names and ID numbers of all books and the duration of the loan in weeks.

```
SELECT Book_id, Book_Name, Max_Time/7  
FROM Books;
```


Arithmetic expressions- example

Books:

<u>Book_Id</u>	Book_Name	Year	Max_Time	Faculty	Pages
1111	Database Systems	1998	7	CS	348
1112	Database Systems	1998	14	CS	348
1113	Database Systems	2001	7	CS	424
2222	Database And Knowledge	1998	1	CS	390
2223	Database And Knowledge	1998	7	EE	390
3333	Electronic Circuits	1998	21	EE	180
4444	Genes 7	1985	7	MED	580
5555	Anatomy	1988	7	MED	450

Arithmetic expressions- example

```
SELECT Book_Id, Book_Name, Max_Time/7  
FROM Books
```

<u>Book_Id</u>	Book_Name	Max_Time
1111	Database Systems	7
1112	Database Systems	14
1113	Database Systems	7
2222	Database And Knowledge	1
2223	Database And Knowledge	7
3333	Electronic Circuits	21
4444	Genes 7	7
5555	Anatomy	7

Where – how to select some record

- The option WHERE <condition> allows to select only part of the records.
- Example: retrieve the names of all books that issued after the year 1990.

```
SELECT Book_Name  
FROM Books  
WHERE Year > 1990;
```

Where – continue

Books:

Book_Id	Book_Name	Year	Max_Time	Faculty	Pages
1111	Database Systems	1998	7	CS	348
1112	Database Systems	1998	14	CS	348
1113	Database Systems	2001	7	CS	424
2222	Database And Knowledge	1998	1	CS	390
2223	Database And Knowledge	1998	7	EE	390
3333	Electronic Circuits	1998	21	EE	180
4444	Genes 7	1985	7	MED	580
5555	Anatomy	1988	7	MED	450

Where – continue

```
SELECT Book_Name  
FROM Books  
WHERE Year > 1990
```

Book_Name
Database Systems
Database Systems
Database Systems
Database And Knowledge
Database And Knowledge
Electronic Circuits

Where – complex conditions

- More complex conditions:
 - Relational operators - $<, >, =, <>, = <, = >$.
 - Logical operators – NOT, OR, AND
- Example: retrieve the names of all the books that issued between 1990 and 2000.

```
SELECT Book_Name
```

```
FROM Books
```

```
WHERE Year => 1990 AND Year <= 2000;
```

Where complex conditions – continue

Books:

Book_Id	Book_Name	Year	Max_Time	Faculty	Pages
1111	Database Systems	1998	7	CS	348
1112	Database Systems	1998	14	CS	348
1113	Database Systems	2001	7	CS	424
2222	Database And Knowledge	1998	1	CS	390
2223	Database And Knowledge	1998	7	EE	390
3333	Electronic Circuits	1998	21	EE	180
4444	Genes 7	1985	7	MED	580
5555	Anatomy	1988	7	MED	450

Where – continue

```
SELECT Book_Name  
FROM Books  
WHERE Year => 1990 AND Year <= 2000
```

Book_Name
Database Systems
Database Systems
Database And Knowledge
Database And Knowledge
Electronic Circuits

Between operator

We can write the previous query in much easier and readable using the BETWEEN operator:

```
SELECT Book_Name  
FROM Books  
WHERE Year BETWEEN 1990 AND 2000;
```

Between operator – another example

- Example: retrieve the names of all books that **not** issued between the years 2000-1990:

```
SELECT Book_Name  
FROM Books  
WHERE NOT (Year BETWEEN 1990 AND 2000)
```

Or..

```
WHERE Year NOT BETWEEN 1990 AND 2000
```

Choosing from a list of values

Example: retrieve all the books that issued in years 1990, 1992, 1999.

```
SELECT Book_Name
```

```
FROM Books
```

```
WHERE Year=1992 OR Year=1998 OR Year=2001;
```

Choosing from a list of values – In operator

We can rewrite the previous query using the IN operator.

- Using IN operator:

```
SELECT Book_Name  
FROM Books  
WHERE Year IN (1992, 1998, 2001);
```

- Using NOT IN:

```
SELECT Book_Name  
FROM Books  
WHERE Year NOT IN (1992, 1998, 2001);
```

Like operator

- The LIKE operator is checking if a string is equal to a given one (aka wildcards).
- The character "_" is referencing a single character.
- The character "%" is reference a series of character in length of 0 or more.
- Example: retrieve all the book names that contains the string Database and the letter before last is m:

```
SELECT Book_Name FROM Books  
WHERE Book_Name LIKE '%Database%m_';
```

Like operator – continue

Books:

Book_Id	Book_Name	Year	Max_Time	Faculty	Pages
1111	Database Systems	1998	7	CS	348
1112	Database Systems	1998	14	CS	348
1113	Database Systems	2001	7	CS	424
2222	Database And Knowledge	1998	1	CS	390
2223	Database And Knowledge	1998	7	EE	390
3333	Electronic Circuits	1998	21	EE	180
4444	Genes 7	1985	7	MED	580
5555	Anatomy	1988	7	MED	450

More about text...

Data bases use a lot of text information. Therefore, the text analysis techniques are blooming.

- Nature Language Processing (NLP)
- Regex
- Tokenizing
- Stemming
- Word embedding
- Entity recognition
- Sentiment analysis

And much more...

Like operator – continue

```
SELECT Book_Name FROM Books  
WHERE Book_Name LIKE '%Database%m_';
```

Book_Name
Database Systems
Database Systems
Database Systems

NULL – missing values

- NULL – a unique value that represent a missing value (“empty cell”)
- For example:
 - The return date of a book that still not returned.
 - A faculty of outside client.
- Compare to NULL:
 - <expr> IS NULL – return true if expr value is NULL.
 - <expr> IS NOT NULL
- Example: retrieve all books that still not returned
SELECT Book_Id FROM Borrowed
WHERE To_Date **IS NULL**;

Distinct – remove duplicate

The operator DISTINCT remove duplicate records from the query result.

Example: retrieve all the book names and issued year without replications.

```
SELECT DISTINCT Book_Name, Year  
FROM Books;
```

Distinct – continue

Books:

Book_Id	Book_Name	Year	Max_Time	Faculty	Pages
1111	Database Systems	1998	7	CS	348
1112	Database Systems	1998	14	CS	348
1113	Database Systems	2001	7	CS	424
2222	Database And Knowledge	1998	1	CS	390
2223	Database And Knowledge	1998	7	EE	390
3333	Electronic Circuits	1998	21	EE	180
4444	Genes 7	1985	7	MED	580
5555	Anatomy	1988	7	MED	450

Distinct – result

Without DISTINCT:

Book_Name	Year
Database Systems	1998
Database Systems	1998
Database Systems	2001
Database And Knowledge	1998
Database And Knowledge	1998
Electronic Circuits	1998
Genes 7	1985
Anatomy	1988

With DISTINCT:

Book_Name	Year
Database Systems	1998
Database Systems	2001
Database And Knowledge	1998
Electronic Circuits	1998
Genes 7	1985
Anatomy	1988

Order By - sorting

ORDER BY – sort the retrieved record by the values of a given attributes or expressions.

Example: retrieve all the book sorted by their issued year.

```
SELECT * FROM Books
```

```
ORDER BY Year;
```

Order By - result

```
SELECT * FROM Books  
ORDER BY Year;
```

Book_Id	Book_Name	Year	Max_Time	Faculty	Pages
4444	Genes 7	1985	7	MED	580
5555	Anatomy	1988	7	MED	450
1111	Database Systems	1998	7	CS	348
1112	Database Systems	1998	14	CS	348
2222	Database And Knowledge	1998	1	CS	390
2223	Database And Knowledge	1998	7	EE	390
3333	Electronic Circuits	1998	21	EE	180
1113	Database Systems	2001	7	CS	424

Desc – decrease sorting

```
SELECT * FROM Books  
ORDER BY Year DESC;
```

Sorting with few attributes



There is a primary sorting and a secondary sorting (like Telephone directory)

Example: retrieve all the books that order by their issued year and by the number of pages(of the same year)

```
SELECT * FROM Books  
ORDER BY Year, Pages;
```


Compute attributes

We can make computing operation on selected attributes

```
SELECT pages*2+5, Book_Name  
FROM Books;
```

AS operator

Operator for naming attributes

```
SELECT pages*2+5 AS Calculated_Pages, Book_Name  
FROM Books
```

Summary functions

Summary functions

In SQL we have the following summary functions:

- MIN – minimum
 - MAX – maximum
 - AVG – average
 - SUM – sum
 - COUNT – count the number of rows.
-
- Each of this function apply on a list of values and return one value.

Summary functions - example

Example: compute the number of the average and maximal pages of all the books.

Solution:

```
SELECT AVG(Pages), MAX(Pages)
FROM Books;
```

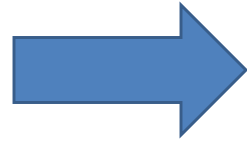
Summary functions - continue

Books:

Book_Id	Book_Name	Year	Max_Time	Faculty	Pages
1111	Database Systems	1998	7	CS	348
1112	Database Systems	1998	14	CS	348
1113	Database Systems	2001	7	CS	424
2222	Database And Knowledge	1998	1	CS	390
2223	Database And Knowledge	1998	7	EE	390
3333	Electronic Circuits	1998	21	EE	180
4444	Genes 7	1985	7	MED	580
5555	Anatomy	1988	7	MED	450

Summary functions - continue

Pages
348
348
424
390
390
180
580
450



AVG(Pages)	MAX(Pages)
389	580

Summary functions - continue

It is possible to compute summary functions only on part of the records by using the **WHERE** option.

For example:

```
SELECT COUNT(Book_Name)
FROM Books
WHERE Year=1998;
```


Summary functions (where) - continue

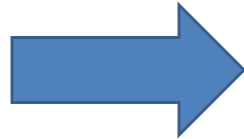
Books:

Book_Id	Book_Name	Year	Max_Time	Faculty	Pages
1111	Database Systems	1998	7	CS	348
1112	Database Systems	1998	14	CS	348
1113	Database Systems	2001	7	CS	424
2222	Database And Knowledge	1998	1	CS	390
2223	Database And Knowledge	1998	7	EE	390
3333	Electronic Circuits	1998	21	EE	180
4444	Genes 7	1985	7	MED	580
5555	Anatomy	1988	7	MED	450

Summary functions (where) - continue

Book_Name
Database Systems
Database Systems
Database Systems
Database And Knowledge
Database And Knowledge
Electronic Circuits
Genes 7
Anatomy

where



Book_Name
Database Systems
Database Systems
Database And Knowledge
Database And Knowledge
Electronic Circuits

Count



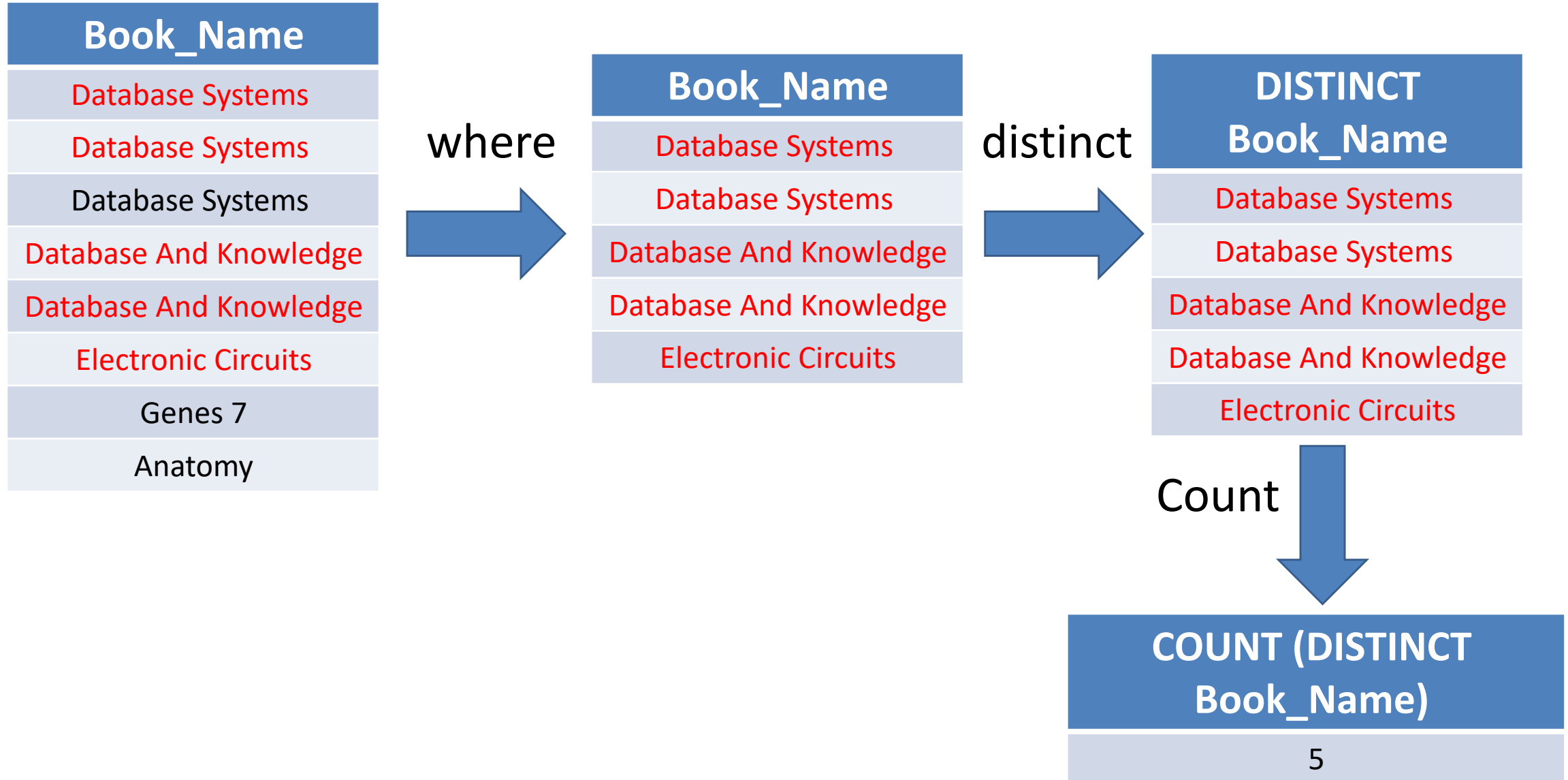
Count Book_Name
5

Summary functions – remove duplicates

- Removing duplicates in time of the summary functions using the DISTINCT operator before the parameter.
- Example:

```
SELECT COUNT (DISTINCT Book_Name)  
FROM Books  
WHERE Year=1998;
```

Summary functions (distinct) - continue



Summary function on null values

- All summary functions are ignoring NULL values.
- Except: COUNT(*)
- Example:

```
SELECT COUNT(*)  
FROM Books  
WHERE Year=1998;
```

Summary functions - continue

- Example: retrieve the number of books each year.

- First try:

```
SELECT Year, COUNT(Book_Id)  
FROM Books;
```

- **Not legal!** (many values of Year, only one value of COUNT).

Grouping functions

Group by

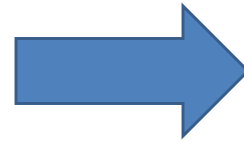
- Apply summary functions on group of records.
- Previous example (fix):

```
SELECT Year, COUNT(Book_Id)  
FROM Books  
GROUP BY Year;
```

- Now each value of year the count is separately computed.

Group by - continue

Book_Id	Year
1111	1998
1112	1998
1113	2001
2222	1998
2223	1998
3333	1998
4444	1985
5555	1988



Year	COUNT (Book_Id)
1998	5
1988	1
2001	1
1985	1

Group by - continue

- Another example:

```
SELECT Faculty, COUNT(Book_Id)
FROM Books
GROUP BY Year;
```

- Not legal! After the grouping by year, in each group could be different faculty values.
- Thumb rule – in addition to the summary operations, we can select attributes that are used in the group operations.

Having – select after grouping

- The HAVING condition – select part of the group that gets from the GROUP BY operation.
- The condition is: summary functions, attributes that used in the GROUP BY operation or expressions.

Having – example

- Example: what the next query returns?

```
SELECT Year, COUNT(Book_Id)
```

```
FROM Books
```

```
GROUP BY Year
```

```
HAVING AVG(Pages) > 400;
```

- Explanation: after the grouping by year, from all the groups we choose those that the average number of pages is greater than 400.

Having – example

Books:

Book_Id	Book_Name	Year	Max_Time	Faculty	Pages
1111	Database Systems	1998	7	CS	348
1112	Database Systems	1998	14	CS	348
1113	Database Systems	2001	7	CS	424
2222	Database And Knowledge	1998	1	CS	390
2223	Database And Knowledge	1998	7	EE	390
3333	Electronic Circuits	1998	21	EE	180
4444	Genes 7	1985	7	MED	580
5555	Anatomy	1988	7	MED	450

Having – example

Year	COUNT(Book_Id)	AVG(Pages)
1998	5	331
2001	1	450
1985	1	424
1988	1	580

AVG(Pages) > 400



Year	COUNT(Book_Id)
2001	1
1985	1
1988	1

Choosing WHERE instead of HAVING

- WHERE: choosing records before the grouping.
- HAVING: choosing groups after the grouping.
- Example: from all of the books with more than 200 pages, compute in each year the number of books that issued, only if the average of the number of pages in that year is greater than 400.

Choosing WHERE instead of HAVING

- Example: from all of the books with more than 200 pages, compute in each year the number of books that issued, only if the average of the number of pages in that year is greater than 400.

```
SELECT Year, COUNT(Book_Id)
FROM Books
WHERE Pages > 200
GROUP BY Year
HAVING AVG(Pages) > 400;
```


Example - continue

Books:

WHERE Pages
> 200

Book_Id	Book_Name	Year	Max_Time	Faculty	Pages
1111	Database Systems	1998	7	CS	348
1112	Database Systems	1998	14	CS	348
1113	Database Systems	2001	7	CS	424
2222	Database And Knowledge	1998	1	CS	390
2223	Database And Knowledge	1998	7	EE	390
3333	Electronic Circuits	1998	21	EE	180
4444	Genes 7	1985	7	MED	580
5555	Anatomy	1988	7	MED	450

Example - continue

- After the "GROUP BY Year":

Year	COUNT(Book_Id)	AVG(Pages)
1998	4	369
2001	1	450
1985	1	424
1988	1	580



HAVING AVG(Pages) > 400

Year	COUNT(Book_Id)
2001	1
1985	1
1988	1

Example - continue

Expiation (stages):

1. Select the records of the books that contain more than 200 pages.
2. Group records into groups, which each group records(books) has the same year.
3. Compute the number of the average number of each group and choosing the groups where their average is greater than 400.
4. Compute the number of books in each group.

Join

Cartesian product

- Syntax: FROM table1, table2, ...
- Meaning: cartesian product between all the tables
- Reference to a field from a certain table: table1.field

```
SELECT Customer.customer_Id, customer_name,  
Book_Name  
FROM Customer, Ordered
```

Join (option A) – Cartesian product with condition

Example: retrieve the names and ID of clients together with the book names they ordered.

Solution:

```
SELECT Customer.Customer_Id, Customer_Name, Book_Name  
FROM Customer, Ordered  
WHERE Customer.Customer_Id = Ordered.Customer_Id
```

Join - continue

Customer_Id	Book_Name	Order_Date
12345	Database Systems	14-Oct-2002
45678	Anatomy	24-Oct-2002
12345	Database And Knowledge	30-Oct-2002
45678	Electronic Circuits	12-Oct-2002

Ordered

Customer_id	Customer_name	Faculty
12345	Moshe Cohen	CS
23456	Avi Barak	EE
34567	Avi Barak	MED
45678	Lior Edri	EE
56789	Moshe Cohen	EE
67890	Moshe Cohen	EE

Customer

Join - result

Customer_id	Customer_name	Book_Name
12345	Moshe Cohen	Database Systems
12345	Moshe Cohen	Database And Knowledge
45678	Lior Edri	Anatomy
45678	Lior Edri	Electronic Circuits

Join – retrieve all attributes from table

Retrieve all attributes of a table

```
SELECT Customer.*, Book_Name  
FROM Customer, Ordered  
WHERE Customer.Customer_Id = Ordered.Customer_Id;
```

Join (option B) – using join operator

Example: retrieve the names and ID of clients together with the book names they ordered.

Solution:

```
SELECT Customer.Customer_Id, Customer_Name, Book_Name  
FROM Customer JOIN Ordered ON  
    Customer.Customer_Id = Ordered.Customer_Id
```

Join (option B) – using join operator

Example: retrieve the names and ID of clients together with the book names they ordered for clients where their name is Cohen.

Solution:

```
SELECT Customer.Customer_Id, Customer_Name, Book_Name  
FROM Customer JOIN Ordered ON  
         Customer.Customer_Id = Ordered.Customer_Id  
WHERE Customer_Name='Choen'
```

Join (option C) – using “using join” operator

Example: retrieve the names and ID of clients together with the book names they ordered for clients where their name is Cohen.

Solution:

```
SELECT Customer.Customer_Id, Customer_Name, Book_Name  
FROM Customer JOIN Ordered USING Customer_Id  
WHERE Customer_Name='Choen';
```

Join (option D) – using natural join operator

Example: retrieve the names and ID of clients together with the book names they ordered for clients where their name is Cohen.

Solution:

```
SELECT Customer.Customer_Id, Customer_Name, Book_Name  
FROM Customer NATURAL JOIN Ordered  
WHERE Customer_Name='Choen';
```

Outer join: retrieve records with
no match on the second table

Target: include all the clients which not ordered any book

Solution:

```
SELECT Customer.Customer_Id, Customer_Name, Book_Name  
FROM Customer LEFT OUTER JOIN Ordered  
ON (Customer.Customer_Id = Ordered.Customer_Id);
```

- Intuition: adding NULL values to the ordered table, that “fits” every client.

Outer join - continue

Result:

Customer_id	Customer_name	Book_Name
12345	Moshe Cohen	Database Systems
12345	Moshe Cohen	Database And Knowledge
45678	Lior Edri	Anatomy
45678	Lior Edri	Electronic Circuits
23456	Avi Barak	
34567	Avi Barak	
56789	Moshe Cohen	
67890	Moshe Cohen	

Another example

Example: What this query do?

```
SELECT Customer.Customer_Id, Customer_Name, COUNT(Book_Name)
FROM Customer LEFT OUTER JOIN Ordered
ON (Customer.Customer_Id = Ordered.Customer_Id)
GROUP BY Customer.Customer_Id, Customer_Name;
```


Another example - continue

Answer: the query return for each client their Id, name, and number of books they ordered.

Results:

Customer_id	Customer_name	COUNT(Book_Name)
12345	Moshe Cohen	2
45678	Lior Edri	2
23456	Avi Barak	0
34567	Avi Barak	0
56789	Moshe Cohen	0
67890	Moshe Cohen	0

What was the result without the outer join?

ALIAS - Synonyms

Goal: another names (usually shorter) to the table

Example:

```
SELECT C.Cust_Id, Cust_Name, Book_Name  
FROM   Customer C, Ordered O  
WHERE  C.Cust_Id = O. Cust_Id;
```

ALIAS – Using the same table

Another use: number of copies of the same table

```
SELECT C1.Cust_Name  
FROM Customer C1, Customer C2  
WHERE C1. Cust_Name = C2. Cust_Name  
      AND  C1.Cust_Id <> C2.Cust_Id;
```

Question: What this query returns?

Answer: all the clients in the library which share the same name with another customer.

This thing is good for reflexive connection!

Group theory operators

We treat a relation as a group of n-items.

Operators in SQL:

- INTERSECT
- MINUS
- UNION

Update

```
UPDATE <tablename>  
SET column-assignment-list  
WHERE conditional-expression;
```

Goal – update values in an existing records on the databases.

Example:

```
UPDATE Books  
SET Max_Time = 7, Faculty='GEN';
```

Insert

```
INSERT INTO <tablename> [(column-list)]  
VALUES (constant-list);
```

Goal: adding new records

Example: adding a new customer to Customer table

```
INSERT INTO Customer  
VALUES (78901, 'Adam Smith', 'CS');
```

Delete

The DELETE operator deletes records from the database. In this command we need to define what records we want to delete.

Example:

```
DELETE FROM Ordered;
```

The table itself is not deleted only the records, meaning the table is now empty.

Subqueries

Subqueries

Motivation: the condition in WHERE can contain an expression which is not previously known, but it depends on the content in the database.

Example: retrieve all the books that were issued in the same year as book number 1112.

```
SELECT Book_Name FROM Books
WHERE Year=
    (SELECT Year
     FROM Books
     WHERE Book_Id = 1112);
```

Subqueries

A subquery can only be in the right position of the compare sign ($>$, $<$, $=$, $<=$, $>=$, $<>$)

Example: The next query is not legal!

```
SELECT Book_Name FROM Books  
WHERE (SELECT Year  
      FROM Books  
      WHERE Book_Id = 1112) = Year;
```

Nested subqueries

We can also use a nested subqueries.

Example: all the faculty that holds books which a client named Lior Edri has ordered.

```
SELECT Faculty FROM Books
WHERE Book_Name IN
(SELECT Book_Name
FROM Ordered
WHERE Customer_Id IN
(SELECT Customer_Id
FROM Customer
WHERE Customer_Name = 'Lior Edri'));
```