Tutorial 3: SQL Queries
Database example

- Library Database

- Customers
  - Cust_Id
  - Cust_Name
  - Faculty

- Ordered
  - Cust_Id
  - Book_Id
  - Order_Date

- Borrowed
  - Cust_Id
  - Book_Id
  - From_Date
  - To_Date

- Books
  - Book_Id
  - Book_Name
  - Year
  - Max_Time
  - Faculty
  - Pages
Database example

- Customers((Cust_Id, Cust_Name, Faculty))
  - Cust_Id: Customer ID (unique)
  - Cust_Name: Customer Name
  - Faculty: Faculty Name
Database example

- Customers(\texttt{Cust\_Id}, \texttt{Cust\_Name}, \texttt{Faculty})

<table>
<thead>
<tr>
<th>Cust_Id</th>
<th>Cust_Name</th>
<th>Faculty</th>
</tr>
</thead>
<tbody>
<tr>
<td>12345</td>
<td>Moshe Cohen</td>
<td>CS</td>
</tr>
<tr>
<td>23456</td>
<td>Avi Barak</td>
<td>EE</td>
</tr>
<tr>
<td>34567</td>
<td>Avi Barak</td>
<td>MED</td>
</tr>
<tr>
<td>45678</td>
<td>Lior Edri</td>
<td>EE</td>
</tr>
<tr>
<td>56789</td>
<td>Moshe Cohen</td>
<td>EE</td>
</tr>
<tr>
<td>67890</td>
<td>Moshe Cohen</td>
<td>EE</td>
</tr>
</tbody>
</table>
Database example

- Books(\texttt{Book\_Id}, \texttt{Book\_Name}, \texttt{Year}, \texttt{Max\_Time}, \texttt{Faculty}, \texttt{Pages})
  - \texttt{Book\_Id}: Unique book id
  - \texttt{Book\_Name}: Bool title
  - \texttt{Year}: Year of print
  - \texttt{Max\_Time}: Maximum borrowing time in days
  - \texttt{Faculty}: Faculty name
  - \texttt{Pages}: Page number
### Database example

- **Books** *(Book_Id, Book_Name, Year, Max_Time, Faculty, Pages)*

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

• Ordered(Cust_Id, Book_Id, Order_Date)
  – Cust_Id: Customer ID
  – Book_Id: Book ID
  – Order_Date: Date of book order
### Database example

- Ordered(\texttt{Cust\_Id, Book\_Id, Order\_Date})

<table>
<thead>
<tr>
<th>Cust_Id</th>
<th>Book_Id</th>
<th>Order_Date</th>
</tr>
</thead>
<tbody>
<tr>
<td>12345</td>
<td>1111</td>
<td>14-Oct-2002</td>
</tr>
<tr>
<td>45678</td>
<td>1112</td>
<td>24-Oct-2002</td>
</tr>
<tr>
<td>12345</td>
<td>1113</td>
<td>30-Oct-2002</td>
</tr>
<tr>
<td>45678</td>
<td>2222</td>
<td>12-Oct-2002</td>
</tr>
</tbody>
</table>
Database example

• Borrowed(\texttt{Book\_Id}, \texttt{Cust\_Id}, \texttt{From\_Date}, \texttt{To\_Date})
  – \texttt{Book\_Id}: Book ID
  – \texttt{Cust\_Id}: Customer ID
  – \texttt{From\_Date}: Borrowing date
  – \texttt{To\_Date}: Return date
Database example

• Borrowed(Book_Id, Cust_Id, From_Date, To_Date)

<table>
<thead>
<tr>
<th>Book_Id</th>
<th>Cust_Id</th>
<th>From_Date</th>
<th>To_Date</th>
</tr>
</thead>
<tbody>
<tr>
<td>5555</td>
<td>56789</td>
<td>13-Oct-2002</td>
<td></td>
</tr>
</tbody>
</table>
Outline

- Preliminaries
  - Relational model definitions
  - Example
- Basic SQL Queries
  - SELECT
  - WHERE
- DB Manipulation
  - DML – DB Data manipulation
  - DDL – DB Definition manipulation
//Choose the data
SELECT [ALL | DISTINCT] {[table.]* | expr [alias], exp [alias], …}

//Data sources
FROM table [alias], table [alias], …

//Condition on the data
[WHERE condition]

//Aggregations
[GROUP BY expr, expr, … [HAVING condition]]

//Groups arithmetics
[{INTERSECT | EXCEPT | UNION | UNION ALL } SELECT …]

//Sorting
[ORDER BY expr [ASC | DESC ], expr [ASC | DESC],…];

• The expressions inside [] are optional
What is the result of the following query?

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

Kahoot

https://play.kahoot.it/#/k/290a440b-b889-4a77-ba12-573531f71d36
SQL Queries: Where - recap

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

<table>
<thead>
<tr>
<th>Book_Id</th>
<th>Book_Name</th>
<th>Year</th>
<th>Max_Time</th>
<th>Pages</th>
<th>Faculty</th>
</tr>
</thead>
<tbody>
<tr>
<td>1111</td>
<td>Database Systems</td>
<td>1998</td>
<td>7</td>
<td>348</td>
<td>CS</td>
</tr>
<tr>
<td>1112</td>
<td>Database Systems</td>
<td>1998</td>
<td>14</td>
<td>348</td>
<td>CS</td>
</tr>
<tr>
<td>1113</td>
<td>Database Systems</td>
<td>2001</td>
<td>7</td>
<td>424</td>
<td>CS</td>
</tr>
<tr>
<td>2222</td>
<td>Database And Knowledge</td>
<td>1998</td>
<td>1</td>
<td>390</td>
<td>CS</td>
</tr>
<tr>
<td>2223</td>
<td>Database And Knowledge</td>
<td>1998</td>
<td>7</td>
<td>390</td>
<td>EE</td>
</tr>
<tr>
<td>3333</td>
<td>Electronic Circuits</td>
<td>1998</td>
<td>21</td>
<td>180</td>
<td>EE</td>
</tr>
<tr>
<td>4444</td>
<td>Genes 7</td>
<td>1985</td>
<td>7</td>
<td>580</td>
<td>MED</td>
</tr>
<tr>
<td>5555</td>
<td>Anatomy</td>
<td>1988</td>
<td>7</td>
<td>450</td>
<td>MED</td>
</tr>
</tbody>
</table>
SELECT Book_Name FROM Books
WHERE Year >= 1990 AND Year <= 2000;

<table>
<thead>
<tr>
<th>Book_Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>Database Systems</td>
</tr>
<tr>
<td>Database Systems</td>
</tr>
<tr>
<td>Database And Knowledge</td>
</tr>
<tr>
<td>Database And Knowledge</td>
</tr>
<tr>
<td>Electronic Circuits</td>
</tr>
</tbody>
</table>
SQL Queries: NULL Values

- NULL – a special value that represents a missing value ("an empty cell").

- Example:
  - The return date of a book that is not yet returned.

- NULL comparison:
  - `expr IS NULL` – returns true if `expr` is NULL value
  - `Expr IS NOT NULL`

- What does the following query return?

  ```sql
  SELECT Book_Id FROM Borrowed WHERE To_Date IS NULL;
  ```

Kahoot
SQL Queries: NULL Values

- **Problem**: the value of an arithmetic or logical expression that contains NULL will always be NULL.
- **Solution**: setting a “default value” for a NULL-able value:
  \[ \text{COALESCE} \text{(value, default)} \]
- Example – Return the borrowing time in weeks for all the books (for not yet returned books, you should return 0).

```sql
SELECT Book_Id, COALESCE(To_Date - From_Date, 0)/7
FROM Borrowed;
```

- In General, `COALESCE(expr1, expr2,...)` returns the first expression that is different then NULL.
SQL Queries: Aggregations

• SQL can compute (aggregate) statistical functions:
  – MIN
  – MAX
  – AVG – average
  – SUM
  – COUNT

• Each of those functions is computes over a set of values and returns (aggregates) a single value

• \( f: P(D) \rightarrow D \)
• Example: Calculate the average and maximal number of pages in books.

```
SELECT AVG(Pages), MAX(Pages)
FROM Books;
```
SELECT \textbf{AVG}(Pages), \textbf{MAX}(Pages) 
FROM books;

<table>
<thead>
<tr>
<th>Book_Id</th>
<th>Book_Name</th>
<th>Year</th>
<th>Max_Time</th>
<th>Pages</th>
<th>Faculty</th>
</tr>
</thead>
<tbody>
<tr>
<td>1111</td>
<td>Database Systems</td>
<td>1998</td>
<td>7</td>
<td>348</td>
<td>CS</td>
</tr>
<tr>
<td>1112</td>
<td>Database Systems</td>
<td>1998</td>
<td>14</td>
<td>348</td>
<td>CS</td>
</tr>
<tr>
<td>1113</td>
<td>Database Systems</td>
<td>2001</td>
<td>7</td>
<td>424</td>
<td>CS</td>
</tr>
<tr>
<td>2222</td>
<td>Database And Knowledge</td>
<td>1998</td>
<td>1</td>
<td>390</td>
<td>CS</td>
</tr>
<tr>
<td>2223</td>
<td>Database And Knowledge</td>
<td>1998</td>
<td>7</td>
<td>390</td>
<td>EE</td>
</tr>
<tr>
<td>3333</td>
<td>Electronic Circuits</td>
<td>1998</td>
<td>21</td>
<td>180</td>
<td>EE</td>
</tr>
<tr>
<td>4444</td>
<td>Genes 7</td>
<td>1985</td>
<td>7</td>
<td>580</td>
<td>MED</td>
</tr>
<tr>
<td>5555</td>
<td>Anatomy</td>
<td>1988</td>
<td>7</td>
<td>450</td>
<td>MED</td>
</tr>
</tbody>
</table>
SELECT AVG(Pages), MAX(Pages)
FROM books;

<table>
<thead>
<tr>
<th>AVG(Pages)</th>
<th>MAX(Pages)</th>
</tr>
</thead>
<tbody>
<tr>
<td>389</td>
<td>580</td>
</tr>
</tbody>
</table>

Pages
- 348
- 348
- 424
- 390
- 390
- 180
- 580
- 450
You can calculate aggregations over a subset of the tuples by using `WHERE`

```sql
SELECT COUNT (Book_Name) 
FROM Books 
WHERE Year = 1998;
```
SQL Queries: Aggregations

SELECT COUNT(Book_name)
FROM Books WHERE Year=1998

<table>
<thead>
<tr>
<th>Book_Id</th>
<th>Book_Name</th>
<th>Year</th>
<th>Max_Time</th>
<th>Pages</th>
<th>Faculty</th>
</tr>
</thead>
<tbody>
<tr>
<td>1111</td>
<td>Database Systems</td>
<td>1998</td>
<td>7</td>
<td>348</td>
<td>CS</td>
</tr>
<tr>
<td>1112</td>
<td>Database Systems</td>
<td>1998</td>
<td>14</td>
<td>348</td>
<td>CS</td>
</tr>
<tr>
<td>1113</td>
<td>Database Systems</td>
<td>2001</td>
<td>7</td>
<td>424</td>
<td>CS</td>
</tr>
<tr>
<td>2222</td>
<td>Database And Knowledge</td>
<td>1998</td>
<td>1</td>
<td>390</td>
<td>CS</td>
</tr>
<tr>
<td>2223</td>
<td>Database And Knowledge</td>
<td>1998</td>
<td>7</td>
<td>390</td>
<td>EE</td>
</tr>
<tr>
<td>3333</td>
<td>Electronic Circuits</td>
<td>1998</td>
<td>21</td>
<td>180</td>
<td>EE</td>
</tr>
<tr>
<td>4444</td>
<td>Genes 7</td>
<td>1985</td>
<td>7</td>
<td>580</td>
<td>MED</td>
</tr>
<tr>
<td>5555</td>
<td>Anatomy</td>
<td>1988</td>
<td>7</td>
<td>450</td>
<td>MED</td>
</tr>
</tbody>
</table>
SELECT COUNT(Book_name) 
FROM Books WHERE Year=1998

<table>
<thead>
<tr>
<th>Book_Name</th>
<th>COUNT</th>
<th>COUNT(Book_Name)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Database Systems</td>
<td></td>
<td>5</td>
</tr>
<tr>
<td>Database Systems</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Database And Knowledge</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Database And Knowledge</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Electronic Circuits</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
• Most of the aggregation functions ignore NULL values
• The exception – COUNT(*)
  – COUNT returns 0 when receiving nothing while
  – other functions returns NULL.
### SQL Queries: Aggregations - NULL

<table>
<thead>
<tr>
<th>Book_Id</th>
<th>Cust_Id</th>
<th>From_Date</th>
<th>To_Date</th>
</tr>
</thead>
<tbody>
<tr>
<td>5555</td>
<td>56789</td>
<td>13-Oct-2002</td>
<td></td>
</tr>
</tbody>
</table>

**Kahoot**

```
SELECT MAX(To_Date) FROM Borrowed;

SELECT COUNT(To_Date) FROM Borrowed;

SELECT COUNT(*) FROM Borrowed;
```
SQL Queries: Aggregations

• Example:
  – Return the number of books printed in each year

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

• **ILLEGAL !!! WRONG!, DON’T DO IT!!!** –
  – There are many values of year, but only value of COUNT
• GROUP BY allows to calculate statistical aggregations over sets of tuples

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

• For each value in Year, the COUNT aggregation is computed separately.
### SQL Queries: Aggregations – GROUP BY

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

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

**Kahoot**
SELECT Year, COUNT(Book_Id) FROM Books GROUP BY Year;
• Another **wrong** example:

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

• **Illegal!** After grouping by Year, in every group there might be different faculty values

• **Rule of thumb:** If you want to select additional fields with your aggregated fields, you must select the fields that you GROUP BY them.
• **HAVING** boolean condition:
  – allows filtering groups that are received from GROUP BY.

• Boolean condition: aggregations, grouped by fields, expressions.
Example: what does the query returns?

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

After grouping by Year, the selected groups are those who got average of pages > 400.
SELECT Year, COUNT(Book_Id) FROM Books
GROUP BY Year
HAVING AVG(Pages) > 400;

<table>
<thead>
<tr>
<th>Book_Id</th>
<th>Book_Name</th>
<th>Year</th>
<th>Max_Time</th>
<th>Pages</th>
<th>Faculty</th>
</tr>
</thead>
<tbody>
<tr>
<td>1111</td>
<td>Database Systems</td>
<td>1998</td>
<td>7</td>
<td>348</td>
<td>CS</td>
</tr>
<tr>
<td>1112</td>
<td>Database Systems</td>
<td>1998</td>
<td>14</td>
<td>348</td>
<td>CS</td>
</tr>
<tr>
<td>1113</td>
<td>Database Systems</td>
<td>2001</td>
<td>7</td>
<td>424</td>
<td>CS</td>
</tr>
<tr>
<td>2222</td>
<td>Database And Knowledge</td>
<td>1998</td>
<td>1</td>
<td>390</td>
<td>CS</td>
</tr>
<tr>
<td>2223</td>
<td>Database And Knowledge</td>
<td>1998</td>
<td>7</td>
<td>390</td>
<td>EE</td>
</tr>
<tr>
<td>3333</td>
<td>Electronic Circuits</td>
<td>1998</td>
<td>21</td>
<td>180</td>
<td>EE</td>
</tr>
<tr>
<td>4444</td>
<td>Genes 7</td>
<td>1985</td>
<td>7</td>
<td>580</td>
<td>MED</td>
</tr>
<tr>
<td>5555</td>
<td>Anatomy</td>
<td>1988</td>
<td>7</td>
<td>450</td>
<td>MED</td>
</tr>
</tbody>
</table>
SQL Queries: Aggregations – HAVING

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

<table>
<thead>
<tr>
<th>Year</th>
<th>COUNT(Book_Id)</th>
<th>AVG(Pages)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1998</td>
<td>5</td>
<td>331</td>
</tr>
<tr>
<td>1988</td>
<td>1</td>
<td>450</td>
</tr>
<tr>
<td>2001</td>
<td>1</td>
<td>424</td>
</tr>
<tr>
<td>1985</td>
<td>1</td>
<td>580</td>
</tr>
</tbody>
</table>

AVG(Pages) > 400

<table>
<thead>
<tr>
<th>Year</th>
<th>COUNT(Book_Id)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1988</td>
<td>1</td>
</tr>
<tr>
<td>2001</td>
<td>1</td>
</tr>
<tr>
<td>1985</td>
<td>1</td>
</tr>
</tbody>
</table>
SQL Queries: Aggregations – HAVING

WHERE: select **tuples** **before** grouping
HAVING: select **groups** **after** grouping

Example:
From the books **with more then 200 pages**, calculate for each publication year the number of books that were printed, given that the **average pages printed in each year is more then 400**.
Example:
From the books with more than 200 pages, calculate for each publication year the number of books that were printed, given that the average pages printed in each year is more than 400.

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

<table>
<thead>
<tr>
<th>Book_Id</th>
<th>Book_Name</th>
<th>Year</th>
<th>Max_Time</th>
<th>Pages</th>
<th>Faculty</th>
</tr>
</thead>
<tbody>
<tr>
<td>1111</td>
<td>Database Systems</td>
<td>1998</td>
<td>7</td>
<td>348</td>
<td>CS</td>
</tr>
<tr>
<td>1112</td>
<td>Database Systems</td>
<td>1998</td>
<td>14</td>
<td>348</td>
<td>CS</td>
</tr>
<tr>
<td>1113</td>
<td>Database Systems</td>
<td>2001</td>
<td>7</td>
<td>424</td>
<td>CS</td>
</tr>
<tr>
<td>2222</td>
<td>Database And Knowledge</td>
<td>1998</td>
<td>1</td>
<td>390</td>
<td>CS</td>
</tr>
<tr>
<td>2223</td>
<td>Database And Knowledge</td>
<td>1998</td>
<td>7</td>
<td>390</td>
<td>EE</td>
</tr>
<tr>
<td>4444</td>
<td>Genes 7</td>
<td>1985</td>
<td>7</td>
<td>580</td>
<td>MED</td>
</tr>
<tr>
<td>5555</td>
<td>Anatomy</td>
<td>1988</td>
<td>7</td>
<td>450</td>
<td>MED</td>
</tr>
</tbody>
</table>
SELECT Year, COUNT(Book_Id) FROM Books
WHERE Pages > 200
GROUP BY Year HAVING AVG(Pages) > 400;

<table>
<thead>
<tr>
<th>Year</th>
<th>COUNT(Book_Id)</th>
<th>AVG(Pages)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1998</td>
<td>4</td>
<td>369</td>
</tr>
<tr>
<td>1988</td>
<td>1</td>
<td>450</td>
</tr>
<tr>
<td>2001</td>
<td>1</td>
<td>424</td>
</tr>
<tr>
<td>1985</td>
<td>1</td>
<td>580</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Year</th>
<th>COUNT(Book_Id)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1988</td>
<td>1</td>
</tr>
<tr>
<td>2001</td>
<td>1</td>
</tr>
<tr>
<td>1985</td>
<td>1</td>
</tr>
</tbody>
</table>
Explanation:
1. WHERE: selection of the books that contains more then 200 pages.
2. GROUP BY: grouping the tuples into groups, so that all the tuples in each group share the same year.
3. Aggregations: calculate the average number of pages in every group, and the number of books in every group.
4. HAVING: choosing the groups where the average page number is at least 400.
Example: what the following query does?

SELECT DISTINCT C1.Cust_Name
FROM Customers C1, Customers C2
WHERE C1.Cust_Name = C2.Cust_Name
    AND C1.Cust_Id <> C2.Cust_Id;

Answer: The names of all the customers such that there is another customer with the same name.
SQL Queries: Subqueries

- Motivation: The WHERE’s Boolean condition result value is unknown and depends on the database content.

- Example: All the books that were printed in the same year as book 1112

```
SELECT Book_Name FROM Books
WHERE Year =
  (SELECT Year FROM Books WHERE Book_Id = 1112);
```
SQL Queries: Subqueries

• Subquery can return multiple values
• You can compare multiple values to multiple result values
• Example: return all the books that were printed in the same year and are in the same faculty as book 1112

```
SELECT Book_Name
FROM Books
WHERE (Year, Faculty) =
    (SELECT Year, Faculty
     FROM Books
     WHERE Book_Id = 1112);
```
SQL Queries: Quantifiers

• Motivation: Comparing a single value to a set of values
• Given the value $x$ and the group $A$ we would like to check if
  – $X$ is bigger than every value in $A$ $\forall a \in A: x > a$
  – $X$ is smaller than some value in $A$ $\exists a \in A: x < a$
  – $x \in A \iff \exists a \in A: x = a$

• SQL quantifiers:
  – $\text{ANY} = \exists$
  – $\text{ALL} = \forall$
• Using quantifiers is done by writing the quantifier after the comparison operator

• Example: All the books with more pages than all the books in ‘CS’ faculty.

```
SELECT Book_Name 
FROM Books 
WHERE Pages > ALL 
  (SELECT Pages FROM Books WHERE Faculty = 'CS');
```
SQL Queries: Quantifiers

- The IN operator purpose is to check if for a value $x$ and a group $A$ $x \in A$
- $\text{IN} \equiv =\text{ANY}$
- $\text{NOT IN} \equiv <> \text{ALL}$
- Example:

```sql
SELECT Book_Name, Year
FROM Books
WHERE Year IN (SELECT Year FROM Books
                WHERE Faculty = 'MED');
```
SQL Queries: Nested Subqueries

• You can nest subqueries
• Example: All the faculties that have books that ordered by a client named “Lior Edri”

```
SELECT Faculty FROM Books
WHERE Book_Id IN
(SELECT Book_Id FROM Ordered
 WHERE Cust_Id IN
 (SELECT Cust_Id FROM Customers
 WHERE Cust_Name = 'Lior Edri'));
```
SQL Queries: Nested Subqueries

• Subqueries vs JOIN
• Let’s write the same query without using nesting

SELECT B.Faculty
FROM Books B, Customers C, Ordered O
WHERE B.Book_Id = O.Book_Id
  AND C.Cust_Id = O.Cust_Id
  AND C.Cust_Name = 'Lior Edri';

• Question: which is more efficient?
• Answer: depends on sever optimization, usually the nested solution is better.
Mutually related subqueries

Motivation: there is a two way connection between the inner and outer queries parameters

Example: return for each faculty the book ids, name and borrowing time for books that can be borrowed longer then the average time period in the same faculty.

```
SELECT Book_Id, Book_Name, Max_Time
FROM Books B
WHERE Max_Time >
    (SELECT AVG(Max_Time) FROM Books
     WHERE Faculty = B.Faculty);
```
SQL Queries: Nested Subqueries

- EXISTS operator is a Boolean operator unary operator which operates on a subquery
- Returns true if the subquery returns something, and false otherwise
- Example: all the clients names which ordered some book

```
SELECT Cust_Name
FROM Customers C
WHERE EXISTS
  (SELECT * FROM Ordered
   WHERE Cust_Id = C.Cust_Id);
```
Example: what the following query does?

SELECT Cust_Name, Customers.Cust_Id, COUNT(Book_Id)
FROM Customers LEFT OUTER JOIN Ordered
    ON (Customers.Cust_Id = Ordered. Cust_Id)
GROUP BY Customers.Cust_Id, Cust_Name;
SQL Queries: OUTER JOIN

```
SELECT Cust_Name, Customers.Cust_Id, COUNT(Book_Id)
FROM Customers
LEFT OUTER JOIN
    Ordered
ON (Customers.Cust_Id = Ordered.Cust_Id)
GROUP BY Customers.Cust_Id, Cust_Name;
```

<table>
<thead>
<tr>
<th>Cust_Id</th>
<th>Cust_Name</th>
<th>Book_Id</th>
</tr>
</thead>
<tbody>
<tr>
<td>12345</td>
<td>Moshe Cohen</td>
<td>1111</td>
</tr>
<tr>
<td>12345</td>
<td>Moshe Cohen</td>
<td>1112</td>
</tr>
<tr>
<td>45678</td>
<td>Lior Edri</td>
<td>1113</td>
</tr>
<tr>
<td>45678</td>
<td>Lior Edri</td>
<td>2222</td>
</tr>
<tr>
<td>23456</td>
<td>Avi Barak</td>
<td></td>
</tr>
<tr>
<td>34567</td>
<td>Avi Barak</td>
<td></td>
</tr>
<tr>
<td>56789</td>
<td>Moshe Cohen</td>
<td></td>
</tr>
<tr>
<td>67890</td>
<td>Moshe Cohen</td>
<td></td>
</tr>
</tbody>
</table>
**SQL Queries: OUTER JOIN**

- **Answer:** return for each client id the client’s name and the number of books they have ordered.

<table>
<thead>
<tr>
<th>Cust_Id</th>
<th>Cust_Name</th>
<th>COUNT (Book_Id)</th>
</tr>
</thead>
<tbody>
<tr>
<td>12345</td>
<td>Moshe Cohen</td>
<td>2</td>
</tr>
<tr>
<td>45678</td>
<td>Lior Edri</td>
<td>2</td>
</tr>
<tr>
<td>23456</td>
<td>Avi Barak</td>
<td>0</td>
</tr>
<tr>
<td>34567</td>
<td>Avi Barak</td>
<td>0</td>
</tr>
<tr>
<td>56789</td>
<td>Moshe Cohen</td>
<td>0</td>
</tr>
<tr>
<td>67890</td>
<td>Moshe Cohen</td>
<td>0</td>
</tr>
</tbody>
</table>

- **What would be the answer without the OUTER JOIN?**

**Kahoot**