Database Management Systems
Course 236363
Faculty of Computer Science
Technion – Israel Institute of Technology

Tutorial 3:
SQL Queries

**Database example**

- Library Database

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

**Database example**

- **Books** *(Book_Id, Book_Name, Year, Max_Time, Faculty, Pages)*
  - Book_Id: Unique book id
  - Book_Name: Book title
  - Year: Year of print
  - Max_Time: Maximum borrowing time in days
  - Faculty: Faculty name
  - Pages: Page number
Database example

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

<table>
<thead>
<tr>
<th>Cust_Id</th>
<th>Book_Id</th>
<th>Order_Date</th>
</tr>
</thead>
<tbody>
<tr>
<td>12346</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(Book_Id, Cust_Id, From_Date, To_Date)
  – Book_Id: Book ID
  – Cust_Id: Customer ID
  – From_Date: Borrowing date
  – To_Date: Return 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

SQL Queries: General Format

//Choose the data
SELECT [ALL | DISTINCT] [[table.| expr [alias], expr [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
SQL Queries: Where - recap

What is the result of the following query?

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

Kahoot

---

SQL Queries: LIKE Operator

- The **LIKE** operator checks for pattern matching between string values and a given pattern (binary operator).
- The character "_" represents a single character (single placeholder)
- The character "%" represents any characters sequence (string) with the length 0 or more.

```sql
SELECT Book_Name FROM Books
WHERE Book_Name LIKE '%Database%' AND Book_Name LIKE 'm_';
```
**SQL Queries: LIKE Operator**

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

**SQL Queries: LIKE Operator**

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

<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>349</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>7</td>
<td>360</td>
<td>CS</td>
</tr>
<tr>
<td>2223</td>
<td>Database And Knowledge</td>
<td>1998</td>
<td>7</td>
<td>360</td>
<td>CS</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>560</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: NULL Values**

- **NULL** – a special values that represents a missing value (“an empty cell”).
- **Example:**
  - The return date of a yet returned book.
- **NULL comparison:**
  - `expr IS NULL` – returns true if `expr` is NULL value
  - `Expr IS NOT NULL`

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

**SQL Queries: NULL Values**

- **A 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: `COALESCE(expr, 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**

- **$f: P(D) \rightarrow D$**

```sql
SELECT AVG(Pages), MAX(Pages) FROM Books;
```
SQL Queries: Aggregations

SELECT AVG(Pages), 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>

• You can calculate aggregations over a subset of the tuples by using WHERE

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>

• Most of the aggregation functions ignore NULL values
• The exception – COUNT(*)

The difference between COUNT and other aggregation functions is that COUNT returns 0 when receiving nothing and other functions will return NULL.

When an aggregation function receives a set of NULL values, it will return NULL (it ignores the NULL values, and receives nothing, so it returns NULL), except COUNT that will return 0, and COUNT(*) that will return the set size.
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 – GROUP BY

• 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

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.
SQL Queries: Aggregations – HAVING

- **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?

```sql
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.

<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>424</td>
</tr>
<tr>
<td>2001</td>
<td>1</td>
<td>390</td>
</tr>
<tr>
<td>1985</td>
<td>1</td>
<td>580</td>
</tr>
</tbody>
</table>

SQL Queries: Aggregations – HAVING

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

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.

```sql
SELECT Year, COUNT(Book_Id) FROM Books
WHERE Pages > 200
GROUP BY Year
HAVING AVG(Pages) > 400;
```
### SQL Queries: Aggregations – HAVING

```sql
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>300</td>
<td>EE</td>
</tr>
<tr>
<td>4444</td>
<td>Genes 7</td>
<td>1985</td>
<td>7</td>
<td>550</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>

WHERE Pages > 200

### Explanation:

1. **WHERE**: selection of the books that contain more than 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.

### SQL Queries: OUTER JOIN

Example: what the following query does?

```sql
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>87890</td>
<td>Moshe Cohen</td>
<td></td>
</tr>
</tbody>
</table>

### Answer:

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

### What would be the answer without the OUTER JOIN?

**Kahoot**
SQL Queries: ALIAS

Example: what the following query does?

```sql
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

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

---

SQL Queries: Quantifiers

• The IN operator purpose is to check if for a value x and a group A x ∈ A
• IN ≡ ANY
• NOT IN ≡ <> 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"

```sql
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

```sql
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.

SQL Queries: Nested Subqueries

- 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.

```sql
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

```sql
SELECT Cust_Name
FROM Customers C
WHERE EXISTS
(SELECT * FROM Ordered
WHERE Cust_Id = C.Cust_Id);
```