Nested Queries
and other bits of SQL

Dr Paolo Guagliardo

University of Edinburgh

Fall 2016
Aggregate results in WHERE
The right way

<table>
<thead>
<tr>
<th>Number</th>
<th>Branch</th>
<th>CustID</th>
<th>Balance</th>
</tr>
</thead>
<tbody>
<tr>
<td>111</td>
<td>London</td>
<td>1</td>
<td>1330.00</td>
</tr>
<tr>
<td>222</td>
<td>London</td>
<td>2</td>
<td>1756.00</td>
</tr>
<tr>
<td>333</td>
<td>Edinburgh</td>
<td>1</td>
<td>450.00</td>
</tr>
</tbody>
</table>

Accounts with a higher balance than the average of all accounts
Aggregate results in **WHERE**

The right way

<table>
<thead>
<tr>
<th>Account</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number</td>
</tr>
<tr>
<td>--------</td>
</tr>
<tr>
<td>111</td>
</tr>
<tr>
<td>222</td>
</tr>
<tr>
<td>333</td>
</tr>
</tbody>
</table>

Accounts with a **higher balance** than the **average of all accounts**

```
SELECT A.number
FROM Account A
WHERE A.balance > ( SELECT AVG(A1.balance) 
                      FROM Account A1 );
```

**Answer:**

<table>
<thead>
<tr>
<th>Number</th>
</tr>
</thead>
<tbody>
<tr>
<td>111</td>
</tr>
<tr>
<td>222</td>
</tr>
</tbody>
</table>
Aggregate results in WHERE

The wrong way

Accounts with a higher balance than the average of all accounts

```
SELECT A.number
FROM Account A
WHERE A.balance > AVG( SELECT A1.balance
FROM Account A1 );
```

ERROR

Aggregate functions can only be used in SELECT and HAVING
Comparisons with subquery results

SELECT ... 
FROM ... 
WHERE term op ( subquery ) ;
Allowed as long as subquery returns a single value

SELECT ... 
FROM ... 
WHERE (term₁, . . ., termₙ) op ( subquery ) ;
Allowed as long as subquery returns a single row with n columns
The **WHERE** clause revisited

term := attribute | value

comparison :=

- \((term, \ldots, term) \ op \ (term, \ldots, term)\)
  with \(op \in \{=, \neq, <, >, \leq, \geq\}\)
- term **IS [NOT] NULL**
- \((term, \ldots, term) \ op \ ANY ( \ query )\)
- \((term, \ldots, term) \ op \ ALL ( \ query )\)
- \((term, \ldots, term) [NOT] IN ( \ query )\)
- **EXISTS** ( \ query )

condition :=

- condition1 **AND** condition2
- condition1 **OR** condition2
- **NOT** condition
Comparisons between tuples

\[(A_1, \ldots A_n) = (B_1, \ldots, B_n) \iff A_1 = B_1 \land \cdots \land A_n = B_n\]

\[(A_1, \ldots A_n) \nless (B_1, \ldots, B_n) \iff A_1 \nless B_1 \lor \cdots \lor A_n \nless B_n\]

\[(A_1, A_2, A_3) < (B_1, B_2, B_n) \quad \text{(generalizes to } n \text{ elements)}\]

\[\iff A_1 < B_1 \lor \left( A_1 = B_1 \land (A_2 < B_2 \lor (A_2 = B_2 \land A_3 < A_3)) \right)\]

\[(A_1, A_2, A_3) \leq (B_1, B_2, B_n) \quad \text{(generalizes to } n \text{ elements)}\]

\[\iff A_1 < B_1 \lor \left( A_1 = B_1 \land (A_2 < B_2 \lor (A_2 = B_2 \land A_3 \leq A_3)) \right)\]
(term, ..., term) \text{ op } \textbf{ANY} (\text{ query })

True if \textbf{there exists} a row \( \bar{r} \) in the results of \text{ query} such that \((\text{term}, \ldots, \text{term}) \text{ op } \bar{r} \) is true

Examples:

\begin{itemize}
  \item 3 < \textbf{ANY}({1, 2, 3}) \text{ is false}
  \item 3 < \textbf{ANY}({2, 3, 4}) \text{ is true}
  \item What about 3 < \textbf{ANY}({}) ?
\end{itemize}
ALL

\[(\text{term, \ldots, term}) \ \text{op} \ \text{ALL} (\ query )\]

True if \textbf{for all} rows \(\bar{r}\) in the results of query \(\ (\text{term, \ldots, term}) \ \text{op} \ \bar{r}\) is true

Examples:

- \(3 < \text{ALL}(\{5, 4, 6\})\) is true
- \(3 < \text{ALL}(\{4, 3, 5\})\) is false
- What about \(3 < \text{ALL}(\{\})\)?
Examples with \texttt{ANY} / \texttt{ALL}

\textbf{ID of customers from London who own an account}

\begin{verbatim}
SELECT  C.custid
FROM    customer C
WHERE   C.city = 'London'
        AND  C.custid = \texttt{ANY} ( SELECT  A.custid
                                     FROM    Account A )
\end{verbatim}
Examples with \texttt{ANY} / \texttt{ALL}

ID of customers from London who own an account

\begin{verbatim}
SELECT C.custid
FROM customer C
WHERE C.city = 'London'
  AND C.custid = \texttt{ANY}( SELECT A.custid
                         FROM Account A );
\end{verbatim}

Customers living in cities without a branch

\begin{verbatim}
SELECT *
FROM customer C
WHERE C.city <> \texttt{ALL} ( SELECT A.branch
                         FROM Account A );
\end{verbatim}
IN / NOT IN

\((\text{term}, \ldots, \text{term}) \ \text{IN} \ (\text{query})\)

same as

\((\text{term}, \ldots, \text{term}) = \text{ANY} \ (\text{query})\)

\((\text{term}, \ldots, \text{term}) \ \text{NOT IN} \ (\text{query})\)

same as

\((\text{term}, \ldots, \text{term}) \leftrightarrow \text{ALL} \ (\text{query})\)
Examples with **IN** / **NOT IN**

ID of customers from London who own an account

```sql
SELECT C.custid
FROM customer C
WHERE C.city = 'London'
AND C.custid IN ( SELECT A.custid
                    FROM Account A );
```

Customers living in cities without a branch

```sql
SELECT *
FROM customer C
WHERE C.city NOT IN ( SELECT A.branch
                        FROM Account A );
```
Examples with **IN / NOT IN**

ID of customers from London who own an account

```sql
SELECT C.custid
FROM customer C
WHERE C.city = 'London'
    AND C.custid IN ( SELECT A.custid
                        FROM Account A );
```

Customers living in cities without a branch

```sql
SELECT *
FROM customer C
WHERE C.city NOT IN ( SELECT A.branch
                        FROM Account A );
```
**EXISTS**

**EXISTS** (query) is true if the result of query is **non-empty**

**(Stupid) Example**

Return all the customers if there are some accounts in London

```
SELECT * 
FROM Customer 
WHERE EXISTS ( SELECT 1 
    FROM Account 
    WHERE branch='London' );
```
Correlated subqueries

All nested queries can refer to attributes in the parent queries

(Smarter) Example

Return customers who have an account in London

```
SELECT * 
FROM Customer C 
WHERE EXISTS ( SELECT 1 
    FROM Account A 
    WHERE branch='London' 
    AND A.custid = C.id );
```

parameters = attributes of a subquery that refer to outer queries
Examples with `EXISTS / NOT EXISTS`

ID of customers from London who own an account

```
SELECT  C.custid
FROM    customer C
WHERE   C.city = 'London'
        AND EXISTS ( SELECT * 
                        FROM   Account A 
                        WHERE  A.custid = C.id );
```
Examples with EXISTS / NOT EXISTS

ID of customers from London who own an account

```
SELECT  C.custid  
FROM    customer C  
WHERE   C.city = 'London'  
        AND EXISTS ( SELECT  *  
                        FROM    Account A  
                        WHERE   A.custid = C.id );
```

Customers living in cities without a branch

```
SELECT  *  
FROM    customer C  
WHERE   NOT EXISTS ( SELECT  *  
                        FROM    Account A  
                        WHERE   A.branch = C.city );
```
Scoping

A subquery has

- a **local scope** (its `FROM` clause)
- \( n \) **outer scopes** (where \( n \) is the **level of nesting**)
  (these are the `FROM` clauses of the parent queries)
Scoping

A subquery has

- a **local scope** (its `FROM` clause)
- **\(n\) outer scopes** (where \(n\) is the level of nesting)
  (these are the `FROM` clauses of the parent queries)

For each **reference** to an attribute

1. Look for a binding in the local scope
2. If no binding is found, look in the **closest** outer scope
3. If no binding is found, look in the next closest outer scope
4. ...
5. If no binding is found, give error
SELECT * 
FROM table1 
WHERE EXISTS ( SELECT 1 
  FROM table2 
  WHERE A = B );

What $A$, $B$ refer to depends on the attributes in `table1` and `table2`
Attribute bindings

```
SELECT * 
FROM table1
WHERE EXISTS ( SELECT 1 
    FROM table2 
    WHERE A = B );
```

What \(A\), \(B\) refer to depends on the attributes in table1 and table2

- Always give aliases to tables
- Always prefix the attributes with the tables they refer to

```
SELECT * 
FROM table1 T1
WHERE EXISTS ( SELECT 1 
    FROM table2 T2 
    WHERE T2.A = T1.B );
```
The **FROM** clause revisited

\[
\text{FROM } \text{table}_1 \left[ \text{AS } T_1 \right], \ldots, \text{table}_n \left[ \text{AS } T_n \right] 
\]

table :=
- base-table
- join-table
- ( query )

join-table :=
- table **JOIN** table **ON** condition
- table **NATURAL JOIN** table
- table **CROSS JOIN** table
Subqueries in FROM

Must always be given a name

```
SELECT * FROM ( SELECT * FROM R )
```

ERROR: subquery in FROM must have an alias
Subqueries in `FROM`

Must always be given a name

```sql
SELECT * FROM ( SELECT * FROM R );
```

ERROR: subquery in FROM must have an alias

Cannot refer to attributes of other tables in the same `FROM` clause

```sql
SELECT *
FROM R, ( SELECT * FROM S WHERE S.a=R.a ) S1 ;
```

ERROR: invalid reference to FROM-clause entry for table "r"
Example: Avoiding **HAVING**

Branches with a total balance (across accounts) of at least 500

```sql
SELECT A.branch
FROM Account A
GROUP BY A.branch
HAVING SUM(A.balance) >= 500 ;
```
Example: Avoiding **HAVING**

Branches with a total balance (across accounts) of at least 500

```
SELECT A.branch
FROM Account A
GROUP BY A.branch
HAVING SUM(A.balance) >= 500 ;
```

Same query without **HAVING**:

```
SELECT subquery.branch
FROM ( SELECT A.branch, SUM(A.balance) AS total
        FROM Account A
        GROUP BY A.branch ) AS subquery
WHERE subquery.total >= 500 ;
```
Example: Aggregation on aggregates

**Average of the total** balances across each customer’s accounts

1. Find the total balance across each customer’s accounts
2. Take the average of the totals

```sql
SELECT AVG(subquery.tot)
FROM ( SELECT A.custid, SUM(A.balance) AS tot
FROM Account A
GROUP BY A.custid ) AS subquery ;
```
Other bits and pieces of SQL
**Ordering**

```
ORDER BY ⟨column₁⟩ [DESC], . . . , ⟨columnₙ⟩ [DESC]
```

Sorts the output rows according to the values of `column₁`
If two rows have the same value for `column₁`, they are sorted by the values of `column₂` and so on . . .

- Default ordering is *ascending* (can be specified with `ASC`)
- *Descending* ordering is specified by `DESC`
Account

<table>
<thead>
<tr>
<th>Number</th>
<th>Branch</th>
<th>CustID</th>
<th>Balance</th>
</tr>
</thead>
<tbody>
<tr>
<td>111</td>
<td>London</td>
<td>1</td>
<td>1330.00</td>
</tr>
<tr>
<td>222</td>
<td>London</td>
<td>2</td>
<td>1756.00</td>
</tr>
<tr>
<td>333</td>
<td>Edinburgh</td>
<td>1</td>
<td>450.00</td>
</tr>
</tbody>
</table>

SELECT *
FROM Account
ORDER BY custid ASC, balance DESC ;
### Ordering example (1)

#### Account

<table>
<thead>
<tr>
<th>Number</th>
<th>Branch</th>
<th>CustID</th>
<th>Balance</th>
</tr>
</thead>
<tbody>
<tr>
<td>111</td>
<td>London</td>
<td>1</td>
<td>1330.00</td>
</tr>
<tr>
<td>222</td>
<td>London</td>
<td>2</td>
<td>1756.00</td>
</tr>
<tr>
<td>333</td>
<td>Edinburgh</td>
<td>1</td>
<td>450.00</td>
</tr>
</tbody>
</table>

```
SELECT *
FROM Account
ORDER BY custid ASC, balance DESC ;
```

<table>
<thead>
<tr>
<th>Number</th>
<th>Branch</th>
<th>CustID</th>
<th>Balance</th>
</tr>
</thead>
<tbody>
<tr>
<td>111</td>
<td>London</td>
<td>1</td>
<td>1330.00</td>
</tr>
<tr>
<td>333</td>
<td>Edinburgh</td>
<td>1</td>
<td>450.00</td>
</tr>
<tr>
<td>222</td>
<td>London</td>
<td>2</td>
<td>1756.00</td>
</tr>
</tbody>
</table>
Ordering example (2)

<table>
<thead>
<tr>
<th>Account</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number</td>
</tr>
<tr>
<td>--------</td>
</tr>
<tr>
<td>111</td>
</tr>
<tr>
<td>222</td>
</tr>
<tr>
<td>333</td>
</tr>
</tbody>
</table>

```
SELECT *
FROM Account
ORDER BY custid DESC, balance ASC ;
```
## Ordering example (2)

### Account

<table>
<thead>
<tr>
<th>Number</th>
<th>Branch</th>
<th>CustID</th>
<th>Balance</th>
</tr>
</thead>
<tbody>
<tr>
<td>111</td>
<td>London</td>
<td>1</td>
<td>1330.00</td>
</tr>
<tr>
<td>222</td>
<td>London</td>
<td>2</td>
<td>1756.00</td>
</tr>
<tr>
<td>333</td>
<td>Edinburgh</td>
<td>1</td>
<td>450.00</td>
</tr>
</tbody>
</table>

```
SELECT *
FROM Account
ORDER BY custid DESC, balance ASC;
```

<table>
<thead>
<tr>
<th>Number</th>
<th>Branch</th>
<th>CustID</th>
<th>Balance</th>
</tr>
</thead>
<tbody>
<tr>
<td>222</td>
<td>London</td>
<td>2</td>
<td>1756.00</td>
</tr>
<tr>
<td>333</td>
<td>Edinburgh</td>
<td>1</td>
<td>450.00</td>
</tr>
<tr>
<td>111</td>
<td>London</td>
<td>1</td>
<td>1330.00</td>
</tr>
</tbody>
</table>
Casting

\[ \text{CAST}( \text{term AS \langle type\rangle} ) \]

Rounding

\[ \text{CAST}(102,4675 \text{ AS NUMERIC}(5,2)) \] gives 102.47
Useful also to produce values in a specific format

Aggregation

\[ \text{AVG( CAST( term AS NUMERIC(p, s) ) )} \]
avoids rounding errors in some systems
Pattern matching

New comparison: term \texttt{LIKE} pattern

where pattern is a string consisting of
characters (case-sensitive!)
- (underscore) wildcard matching any one character
% (percent) wildcard matching any substring (including empty)

Example
Customers with a name that \textit{begins with 'K'}
and has \textit{at least 5 characters}

\begin{verbatim}
SELECT * 
FROM Customer
WHERE name \texttt{LIKE 'K____%'} ;
\end{verbatim}