# Introduction to Databases

(INFR10080)

## (Nested Queries)

Instructor: Yang Cao

(Fall 2025)

THE UNIVERSITY *of* EDINBURGH

## Changelog

v25.0 Initial version

# Aggregate results in WHERE

**Account**

| Number | Branch | CustID | Balance |
|---|---|---|---|
| 111 | London | 1 | 1330.00 |
| 222 | London | 2 | 1756.00 |
| 333 | Edinburgh | 1 | 450.00 |

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:

| Number |
|---|
| 111 |
| 222 |



# Aggregate results in WHERE

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 (term$_1$, ..., term$_n$) **op** ( subquery ) ;

Allowed as long as subquery returns a **single row** with *n* columns



# The WHERE clause revisited

term := attribute | value

comparison :=

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

condition :=

- comparison
- condition AND condition
- condition OR condition
- NOT condition

# Comparisons between tuples

## Equality

$$(t_1, \ldots t_n) = (t'_1, \ldots, t'_n) \iff t_1 = t'_1 \text{ AND } \cdots \text{ AND } t_n = t'_n$$

## Less-than

$$(t_1, t_2, \ldots, t_n) < (t'_1, t'_2, \ldots, t'_n) \iff$$
$$t_1 < t'_1 \text{ OR } \left(t_1 = t'_1 \text{ AND } (t_2, \ldots, t_n) < (t'_2, \ldots, t'_n)\right)$$

## Others (derived)

For $\bar{t} = (t_1, \ldots, t_n)$ and $\bar{t}' = (t'_1, \ldots, t'_n)$:

$$\begin{array}{lcl}
\bar{t} <> \bar{t}' & \iff & \text{NOT } (\bar{t} = \bar{t}') \\
\bar{t} <= \bar{t}' & \iff & \bar{t} < \bar{t}' \text{ OR } \bar{t} = \bar{t}' \\
\bar{t} > \bar{t}' & \iff & \text{NOT } (\bar{t} <= \bar{t}') \\
\bar{t} >= \bar{t}' & \iff & \text{NOT } (\bar{t} < \bar{t}')
\end{array}$$



# ANY

(term, . . . , term) **op** ANY ( query )

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

Examples:

Consider the table T =

| A |
|---|
| 1 |
| 2 |
| 3 |

- $3 <$ ANY(SELECT A FROM T) is false
- $3 <$ ANY(SELECT A+1 FROM T) is true
- What about $3 <$ ANY(SELECT A FROM T WHERE A $= 0$) ?

# ALL

(term, . . . , term) **op** ALL ( query )

True if **for all** rows $\bar{r}$ in the results of query
(term, . . . , term) **op** $\bar{r}$ is true

Examples:

Consider the table T =

| A |
|---|
| 3 |
| 4 |
| 5 |
| 6 |

- $3 <$ ALL(SELECT A FROM T WHERE A <> 3) is true
- $3 <$ ALL(SELECT A FROM T WHERE A <> 6) is false
- What about $3 <$ ALL(SELECT A FROM T WHERE A $= 0$) ?



# Examples with ANY / ALL

Customer: *ID, Name, City*
Account: *Number, Branch, CustID, Balance*

ID of customers from London who own an account

```
SELECT C.id
FROM   Customer C
WHERE  C.city = 'London'
  AND  C.id = ANY( SELECT A.custid
                   FROM   Account A );
```

Customers living in cities without a branch

```
SELECT *
FROM   Customer C
WHERE  C.city <> ALL ( SELECT A.branch
                       FROM   Account A );
```

# IN / NOT IN

(term, . . . , term) IN ( query )

same as

(term, . . . , term) = ANY ( query )

(term, . . . , term) NOT IN ( query )

same as

(term, . . . , term) <> ALL ( query )



# Examples with IN / NOT IN

ID of customers from London who own an account

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

Customers living in cities without a branch

```
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  A.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.id
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)

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

# 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

FROM table$_1$ [ [ AS ] $T_1$ ], . . . , table$_n$ [ [ AS ] $T_n$ ]

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

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

```
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

```
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

```
SELECT    AVG(subquery.tot)
FROM      ( SELECT   A.custid, SUM(A.balance) AS tot
            FROM     Account A
            GROUP BY A.custid ) AS subquery ;
```