

SQL Database Manipulations: SELECT statements

Thomas Schwarz, SJ

SELECT

- SELECT is the most frequent command
 - Basic use:
 - SELECT attribute1, attribute2, ... FROM databasetable
 - SELECT * FROM databasetable

SELECT

- SELECT — WHERE clause:
 - Imposes a condition on the results

SELECT

- = equals (comparison operator)
- AND, OR
- IN, NOT IN
- LIKE, NOT LIKE
- BETWEEN ... AND
- EXISTS, NOT EXISTS
- IS NULL, IS NOT NULL
- comparison operators

SELECT

- AND operator
 - Combines two statements (concerning one or more tables)

```
SELECT
```

```
    *
```

```
FROM
```

```
    employees
```

```
WHERE
```

```
    first_name = 'Denis' and gender = 'M';
```

SELECT

- OR is the Boolean or
- Trick Question: How many records will this query return?

```
SELECT
    *
FROM
    employees
WHERE
    last_name = 'Denis' AND gender = 'M' OR gender = 'F'
```

SELECT

- Operator precedence:
 - AND < OR

```
SELECT
    *
FROM
    employees
WHERE
    last_name = 'Denis' AND (gender = 'M' OR gender = 'F')
```

SELECT

- Quiz:
 - Retrieve all female employees with first name 'Aruna' or 'Kelly'


```
SELECT
    *
FROM
    employees
WHERE
    gender = 'F' AND
    (first_name = 'Aruna' OR first_name = 'Kelly');
```

SELECT

- IN, NOT IN
 - Checks for membership in lists
 - MySQL: faster than equivalent OR formulation

```
SELECT
    *
FROM
    employees
WHERE
    first_name NOT IN ('Elvis', 'Kevin', 'Thomas');
```

SELECT

- LIKE
 - Pattern matching
 - Wild cards
 - % means zero or more characters
 - _ means a single letter
 - [] means any single character within the bracket
 - ^ means any character not in the bracket
 - - means a range of characters

Like Examples

- WHERE name LIKE 't%'
 - any values that start with 't'
- WHERE name LIKE '%t'
 - any values that end with 't'
- WHERE name LIKE '%t%'
 - any value with a 't' in it
- WHERE name LIKE '_t%'
 - any value with a 't' in second position

Like Examples

- WHERE name LIKE '[ts]%'
 - any values that start with 't' or 's'
- WHERE name LIKE '[t-z]%'
 - any values that start with 't', 'u', 'v', 'w', 'x', 'y', 'z'
- WHERE name LIKE '[!ts]%'
 - any value that does not start with a 't' or a 's'
- WHERE name LIKE '_t%'
 - any value with a 't' in second position

SELECT

- BETWEEN ... AND ...
 - Selects records with a value in the range
 - endpoints included

```
SELECT
    *
FROM
    employees
WHERE
    hire_data between 1990-01-01 and 1999-12-31;
```

SELECT

- SELECT DISTINCT

```
SELECT DISTINCT
    gender
FROM
    employees
```

SELECT

- Aggregate Functions
 - Applied to a row of a result table
 - COUNT
 - SUM
 - MIN
 - MAX
 - AVG

SELECT

- SELECT COUNT

- ```
SELECT
 COUNT (emp_no)
FROM
 employees
```

# SELECT

- SELECT COUNT

```
SELECT COUNT(employees.emp_no)
FROM employees
WHERE
 first_name LIKE ('Tom%') or first_name
LIKE ('Tho%');
```

# SELECT

- Combine COUNT with DISTINCT

```
SELECT
 COUNT(DISTINCT first_name, last_name)
FROM
 employees
```

# SELECT

- Combine COUNT with DISTINCT

```
SELECT
 COUNT(DISTINCT emp_no)
FROM
 salaries
WHERE
 salary >=100000;
```

# SELECT

- ORDER BY
  - Orders result by default in ascending order
    - ASC ascending
    - DSC descending

```
SELECT
 *
FROM
 employees
WHERE
 hire_date > '2000-01-01'
ORDER BY first_name;
```

# SELECT

- GROUP BY
  - Just before ORDER BY in a query
  - Needed with aggregate functions
  - Example: Getting all first names in order

```
SELECT
 first_name
FROM
 employees
GROUP BY first_name;
```

# SELECT

- GROUP BY
  - Example: Counting first names in the employee data base
  - Hint: you want to include the attribute on which you group

```
SELECT
 first_name, COUNT(first_name)
FROM
 employees
GROUP BY first_name
ORDER BY first_name;
```

# SELECT

- GROUP BY
  - Example: Counting first names in the employee data base
    - To make it look better, add an AS clause

```
SELECT
 first_name, COUNT(first_name)
FROM
 employees
GROUP BY first_name
ORDER BY first_name;
```



# In Class Exercises

- Using MySQL Workbench
  - Create a new database called TEST
  - Create a table R with attributes A and B of type INT
  - Insert these values into R using insert statements such as `INSERT INTO R(A,B) VALUES(3,9);`
  - Use a `SELECT` statement to insure that the table is correct (including the double values)

| A | B |
|---|---|
| 1 | 2 |
| 1 | 3 |
| 1 | 4 |
| 2 | 1 |
| 2 | 3 |
| 3 | 1 |
| 3 | 2 |
| 3 | 9 |
| 4 | 2 |
| 4 | 2 |

# In Class Exercises

- Obtain a table that lists the average value of B (AVG) for all values of A

| A | B |
|---|---|
| 1 | 2 |
| 1 | 3 |
| 1 | 4 |
| 2 | 1 |
| 2 | 3 |
| 3 | 1 |
| 3 | 2 |
| 3 | 9 |
| 4 | 2 |
| 4 | 2 |

| A | BAve |
|---|------|
| 1 | 3.0  |
| 2 | 2.0  |
| 3 | 4.0  |
| 4 | 2.0  |

# In Class Exercises

```
SELECT
 A, AVG(B) as BAve
FROM
 R
GROUP BY A;
```

# In Class Exercises

- Obtain the same table, but in descending order of A

```
SELECT
 A, AVG(B) AS bAve
FROM
 R
GROUP BY A
ORDER BY A DESC;
```

# In Class Exercises

- Create a table that contains only the unique value pairs for A and B

# In Class Exercises

```
SELECT DISTINCT
 *
FROM
 R;
```

# In Class Exercises

- How many entries does the table have with and without uniqueness constraints?

# In Class Exercises

```
SELECT
 COUNT (A,B) AS numberOfRecords
FROM
 R;
```

```
SELECT
 COUNT (DISTINCT A,B) AS numberOfRecords
FROM
 R;
```



# In Class Exercises

- Find the average and the number of counts for all B-values depending on the A-value

| A | countb | aveB   |
|---|--------|--------|
| 1 | 3      | 3.0000 |
| 2 | 2      | 2.0000 |
| 3 | 3      | 4.0000 |
| 4 | 2      | 2.0000 |

# In Class Exercises

```
SELECT
 A, COUNT(B) AS countb, AVG(B) AS aveB
FROM
 R
```

| A | countb | aveB   |
|---|--------|--------|
| 1 | 3      | 3.0000 |
| 2 | 2      | 2.0000 |
| 3 | 3      | 4.0000 |
| 4 | 2      | 2.0000 |

# In Class Exercises

- Do the same, but make sure that we do not count double rows twice

# In Class Exercises

```
SELECT
 A, COUNT(B) AS countb, AVG(B) AS aveB
FROM (
 SELECT DISTINCT
 A, B
 FROM
 R
) AS AUnique
GROUP BY A;
```

| A | countb | aveB   |
|---|--------|--------|
| 1 | 3      | 3.0000 |
| 2 | 2      | 2.0000 |
| 3 | 3      | 4.0000 |
| 4 | 1      | 2.0000 |

# In Class Exercises

- Select the count of B-values and average of B-values where the A value is at least 3
  - We modify this with a WHERE clause
  - The WHERE is applied to all tuples first, then the grouping and the calculation of the aggregate function happens

# In Class Exercises

```
SELECT
 A, COUNT(B) AS countb, AVG(B) AS aveB
FROM
 (SELECT DISTINCT
 A, B
 FROM
 R) AS AUnique
WHERE
 A > 2
GROUP BY A;
```

| A | countb | aveB   |
|---|--------|--------|
| 3 | 3      | 4.0000 |
| 4 | 1      | 2.0000 |

# Having

- A WHERE clause applies to all the rows, but it cannot apply to the groups created by the GROUP BY
  - For this, SQL introduces the HAVING clause
  - Just like a WHERE clause, but refers to aggregated data

# Having

- Syntax of Having

```
SELECT column_name(s)
FROM table_name
WHERE condition
GROUP BY column_name(s)
HAVING condition
ORDER BY column_name(s) ;
```



# Having

- Difference between WHERE and HAVING
  - WHERE is only for selecting tuples
  - HAVING can only refer to the group-by-ed attribute

# In Class Exercises

- Insert another double tuple 1, 1
- Get count and average of the B-values in dependence on A where the count is 2 or less

Table 1

| A | B |
|---|---|
| 1 | 2 |
| 1 | 3 |
| 1 | 4 |
| 2 | 1 |
| 2 | 3 |
| 3 | 1 |
| 3 | 2 |
| 3 | 9 |
| 4 | 2 |
| 4 | 2 |
| 1 | 1 |
| 1 | 1 |

# In Class Exercises

```
SELECT
 A, COUNT (B) , AVG (B)
FROM
 R
GROUP BY A
HAVING COUNT (B) <= 2;
```

Table

| A | B |
|---|---|
| 1 | 2 |
| 1 | 3 |
| 1 | 4 |
| 2 | 1 |
| 2 | 3 |
| 3 | 1 |
| 3 | 2 |
| 3 | 9 |
| 4 | 2 |
| 4 | 2 |
| 1 | 1 |
| 1 | 1 |

Table 1

| A | COUNT(B) | AVG(B) |
|---|----------|--------|
| 2 | 2        | 2.0000 |
| 4 | 2        | 2.0000 |

# In Class Exercises

- Get count and average of the B-values in dependence on A where A is less than or equal to 2

# In Class Exercises

```
SELECT
 A, COUNT (B) , AVG (B)
FROM
 R
WHERE
 A <= 2
GROUP BY A;
```

Table 1

| A | B |
|---|---|
| 1 | 2 |
| 1 | 3 |
| 1 | 4 |
| 2 | 1 |
| 2 | 3 |
| 3 | 1 |
| 3 | 2 |
| 3 | 9 |
| 4 | 2 |
| 4 | 2 |
| 1 | 1 |
| 1 | 1 |

Table 1-1

| A | COUNT(B) | AVG(B) |
|---|----------|--------|
| 1 | 5        | 2.2000 |
| 2 | 2        | 2.0000 |

# SELECT

- LIMIT gives the maximum number of rows returned
  - Can be used for a sample
  - Can be used with ORDER BY ASC

# SELECT

- Use the employees database
  - Find the five employees that have made the most money
    - Hint: The Salary table has the information but employees have different salaries over time

# SELECT

```
SELECT
 first_name, last_name, MAX(salary)
FROM
 salaries,
 employees
WHERE
 employees.emp_no = salaries.emp_no
GROUP BY salaries.emp_no
ORDER BY MAX(salary) DESC
LIMIT 5;
```

Table 1

| first_name      | last_name | MAX(salary) |
|-----------------|-----------|-------------|
| <b>Tokuyasu</b> | Pesch     | 158220      |
| <b>Xiahua</b>   | Whitcomb  | 155709      |
| <b>Tsutomu</b>  | Alameldin | 155377      |
| <b>Willard</b>  | Baca      | 154459      |
| <b>Ibibia</b>   | Junet     | 150345      |