SQL – Advanced Topics

Contents

1. Aggregating Data
    - Group Functions and GROUP BY
2. MySQL SQL Functions
3. MySQL Data Types
4. Data Definition Language (DDL)
5. Creating Tables in MySQL
6. Inserting / Updating / Deleting Data

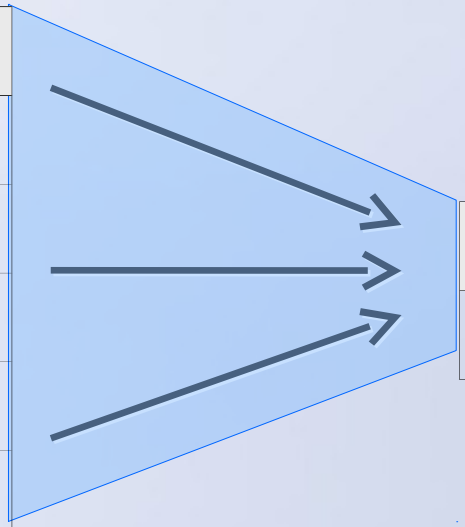
SQL Language

Aggregating Data

Group Functions

- Group functions operate on sets of rows to give one result per group

EMPLOYEE_ID	SALARY
100	24000
101	17000
102	17000
103	9000
104	6000
...	...



MAX(SALARY)
24000

Group Functions in SQL

- `COUNT(*)` – count of the selected rows
- `SUM(column)` – sum of the values in given column from the selected rows
- `AVG(column)` – average of the values in given column
- `MAX(column)` – the maximal value in given column
- `MIN(column)` – the minimal value in given column

AVG() and SUM() Functions

- You can use AVG() and SUM() for numeric data types

```
SELECT AVG(SALARY), MAX(SALARY),
  MIN(SALARY), SUM(SALARY)
FROM employees
WHERE JOB_ID LIKE '%REP%'
```

AVG(SALARY)	MAX(SALARY)	MIN(SALARY)	SUM(SALARY)
8272.72	11500	6000	273000

MIN() and MAX() Functions

- You can use MIN() and MAX() for any data type (number, date, varchar, ...)

```
SELECT MIN(HIRE_DATE), MAX(HIRE_DATE)
FROM employees
```

MIN(HIRE_DATE)	MAX(HIRE_DATE)
17-JUN-1987	29-JAN-00

- Displaying the first and last employee's name in alphabetical order:

```
SELECT MIN(LAST_NAME), MAX(LAST_NAME)
FROM employees
```

The COUNT(...) Function

- COUNT(*) returns the number of rows in the result table

```
SELECT COUNT(*) FROM employees
WHERE DEPARTMENT_ID = 50
```

COUNT(*)
5

- COUNT(*expr*) returns the number of rows with non-null values for the *expr*

```
SELECT COUNT(COMMISSION_PCT)
FROM employees
WHERE DEPARTMENT_ID = 80
```

COUNT(COMMISION_PCT)
3

- COUNT(DISTINCT *expr*) returns the number of distinct non-null values

Group Functions and Nulls

- Group functions ignore all `null` values in the column

```
SELECT AVG(COMMISSION_PCT) FROM employees
```

AVG(COMMISSION_PCT)
.2229

- If each null value in `COMMISSION_PCT` is considered as `0` and is included in the calculation, the result would be `0.0425`

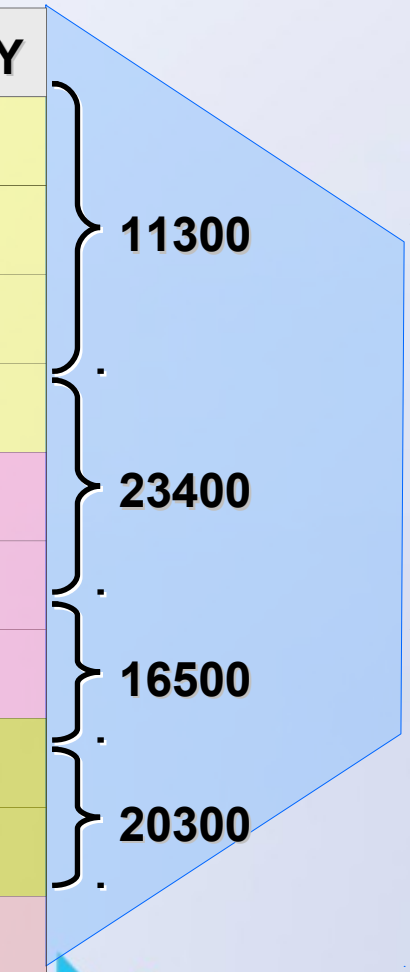
SQL Language

Group Functions and the `GROUP BY` Statement

Creating Groups of Data

`employees`

DEPARTMENT_ID	SALARY
50	3100
50	3000
50	2600
50	2600
20	4400
20	13000
20	6000
40	6500
40	10000
110	12000
110	8300
...	...



DEPARTMENT_ID	SUM(SALARY)
50	11300
20	23400
40	16500
110	20300
...	...

The GROUP BY Statement

- We can divide rows in a table into smaller groups by using the GROUP BY clause
- The syntax:

```
SELECT <columns>, <group_function(column)>
FROM   <table>
[WHERE <condition>]
[GROUP BY  <group_by_expression>]
[ORDER BY  <columns>
```

- The *<group_by_expression>* is a list of columns

The GROUP BY Statement

- Example of grouping data:

```
SELECT DEPARTMENT_ID, SUM(SALARY)
FROM employees
GROUP BY DEPARTMENT_ID
```

DEPARTMENT_ID	SUM(SALARY)
100	51600
30	24900
(null)	7000
...	...

- The GROUP BY column is not required to be in the SELECT list

Grouping by Several Columns

employees

DEPARTMENT_ID	JOB_ID	SALARY
20	AD_ASST	4400
20	MK_MAN	13000
20	MK_MAN	12000
30	PU_CLERK	2500
30	PU_CLERK	2500
30	PU_CLERK	2500
30	PU_MAN	11000
30	PU_MAN	11500
30	PU_MAN	10000
30	PU_MAN	11000
...	...	...

4400

25000

7500

43500

employees

DPT_ID	JOB_ID	SUM(SALARY)
20	AD_ASST	4400
20	MK_MAN	25000
30	PU_CLERK	7500
30	PU_MAN	43500
...	...	...

Grouping by Several Columns – Example

- Example of grouping data by several columns:

```
SELECT DEPARTMENT_ID, JOB_ID,
  COUNT(EMPLOYEE_ID), SUM(SALARY)
FROM employees
GROUP BY DEPARTMENT_ID, JOB_ID
ORDER BY SUM(SALARY) DESC
```

DEPARTMENT_ID	JOB_ID	COUNT(EMPLOYEE_ID)	SUM(SALARY)
80	SA_REP	29	243500
50	SH_CLERK	20	64300
80	SA_MAN	5	61000
…	…	…	…

Illegal Queries

- This `SELECT` statement is illegal

```
SELECT DEPARTMENT_ID, COUNT(LAST_NAME)
FROM employees
```

  - Can not combine columns with groups functions unless when using `GROUP BY`

- This `SELECT` statement is also illegal

```
SELECT DEPARTMENT_ID, AVG(SALARY)
FROM employees
WHERE AVG(SALARY) > 8000
GROUP BY DEPARTMENT_ID;
```

  - Can not use `WHERE` for group functions

Using GROUP BY with HAVING Clause

- HAVING works exactly like WHERE but is used for the grouping functions

```
SELECT DEPARTMENT_ID,
  COUNT(EMPLOYEE_ID), AVG(SALARY)
FROM employees
GROUP BY DEPARTMENT_ID
HAVING COUNT(EMPLOYEE_ID) BETWEEN 3 AND 6
```

DEPARTMENT_ID	COUNT(EMPLOYEE_ID)	AVG(SALARY)
100	6	8600
30	6	4150
90	3	19333.33
60	5	5760

Using Grouping Functions and Table Joins

- We can apply grouping function from joined tables

```
SELECT COUNT(*) AS EMPS, DEPARTMENT_NAME
FROM employees E JOIN departments D
  ON E.DEPARTMENT_ID=D.DEPARTMENT_ID
WHERE
  HIRE_DATE BETWEEN '1991-1-1' AND '1997-12-31'
GROUP BY DEPARTMENT_NAME
HAVING COUNT(*) > 5
ORDER BY EMPS DESC
```

EMPS	DEPARTMENT_NAME
19	Shipping
15	Sales

Using Grouping Functions and Table Joins (2)

- All selected columns should stay in the `GROUP BY` statement or in a group function

```
SELECT D.DEPARTMENT_ID, D.DEPARTMENT_NAME, L.CITY,
  MAX(E.SALARY) AS MAX_SALARY
FROM employees E JOIN departments D
  ON E.DEPARTMENT_ID=D.DEPARTMENT_ID
  JOIN locations L
  ON L.LOCATION_ID = D.LOCATION_ID
GROUP BY D.DEPARTMENT_ID, D.DEPARTMENT_NAME,
  L.LOCATION_ID, L.CITY
ORDER BY MAX_SALARY DESC
```

DEPARTMENT_ID	DEPARTMEN_NAME	CITY	MAX_SALARY
90	Executive	Seattle	24000
80	Sales	Oxford	14000

SQL Language

MySQL Functions

Standard Functions in MySQL

- Single-row functions
  - String functions
  - Mathematical functions
  - Date functions
  - Conversion functions
- Multiple-row functions
  - Group functions

COALESCE() Function

COALESCE(<value>,<value>, <value>)

– returns first non-NULL value

```
SELECT
 CITY, COALESCE(STATE_PROVINCE, 'N/A') AS
PROVINCE FROM LOCATIONS
```

CITY	PROVINCE
Venice	N/A
Tokyo	Tokyo Prefecture
Hiroshima	N/A
Southlake	Texas
South San Francisco	California
...	...

IF() Function

IF(<condition>, <true-res>,<false-res>) – returns one or another value based on condition

```
SELECT FIRST_NAME, LAST_NAME,
 IF(MANAGER_ID IS NULL, 'The big boss',
 'Just an employee') as POSITION
FROM EMPLOYEES
```

FIRST_NAME	LAST_NAME	POSITION
Steven	King	The big boss
Neena	Kochhar	Managed directly by the big boss
...	...	...

String Functions

- Changing the casing – `LOWER()`, `UPPER()`, `INITCAP()`
- Manipulating characters – `CONCAT()`, `SUBSTR()`, `LENGTH()`, `INSTR()`, `LPAD()`, `RPAD()`, `TRIM()`, `REPLACE()`

```
SELECT LAST_NAME, LENGTH(LAST_NAME)
FROM employees
WHERE UPPER(LAST_NAME) LIKE 'LA%'
```

LAST_NAME	LENGTH(LAST_NAME)
Ladwig	6
Landry	6

Date Functions

SELECT NOW() -> return current date and time

SELECT something FROM tbl_name WHERE
DATE_SUB(CURDATE(),INTERVAL 30 DAY) <= date_col;
→ Return something in the last 30 days

SELECT DAYOFMONTH('2001-11-05'),
MONTH('2005-09-12');
→ 5,9

SELECT DATE_ADD('2008-01-02', INTERVAL 31 DAY);
-> '2008-02-02‘

SELECT CONVERT_TZ('2004-01-01
12:00:00','GMT','MET');
-> '2004-01-01 13:00:00'

Other Functions

- Mathematical Functions – FLOOR(), CEIL(), MOD()

```
SELECT FLOOR(3.14) → 3
SELECT CEIL(5.86) → 6
```

- Conversion Functions – STR_TO_DATE(<value>, <format>)

```
SELECT STR_TO_DATE('01,5,2013','%d,%m,%Y');
→ '2013-05-01'
```

SQL Language

Data Definition Language (DDL)

Data Definition Language

- Types of commands
  - Defining / editing objects
    - `CREATE`
    - `ALTER`
    - `DROP`
  - Managing access permissions
    - `GRANT`
    - `REVOKE`

Creating Objects

- **CREATE / CREATE OR REPLACE commands**
  - **CREATE TABLE <name> (<fields definitions>)**
  - **CREATE SEQUENCE <name>**
  - **CREATE VIEW <name> AS <select>**

```
CREATE TABLE PERSONS (
   PERSON_ID INTEGER NOT NULL,
   NAME NVARCHAR2(50) NOT NULL,
   CONSTRAINT PERSON_PK PRIMARY KEY(PERSON_ID)
)

CREATE OR REPLACE VIEW PERSONS_TOP_10 AS
SELECT NAME FROM PERSONS LIMIT 10
```

Database Views

- Views are named SQL SELECT queries
  - Usually views join multiple tables and provide filtering
  - Views simplify queries

```
CREATE VIEW V_SEATTLE_employees AS
  SELECT E.EMPLOYEE_ID, E.FIRST_NAME,
    E.LAST_NAME, CITY, DEPARTMENT_NAME
  FROM employees E
  JOIN departments D
    ON D.DEPARTMENT_ID = E.DEPARTMENT_ID
  JOIN locations L
    ON D.LOCATION_ID = L.LOCATION_ID
  WHERE CITY='Seattle'
```

Modifying Objects

- **ALTER command**
  - **ALTER TABLE <name> <command>**
  - **ALTER**

```
-- Add a foreign key constraint TOWN --> COUNTIRY
ALTER TABLE TOWN
  ADD CONSTRAINT TOWN_COUNTRY_FK
  FOREIGN KEY (COUNTRY_ID)
  REFERENCES COUNTRY(ID) ENABLE

-- Add column COMMENT to the table PERSON
ALTER TABLE PERSONS ADD ("COMMENT" VARCHAR2(800))

-- Remove column COMMENT from the table PERSON
ALTER TABLE PERSONS DROP COLUMN "COMMENT"
```

Deleting Objects

- DROP command
  - DROP TABLE <name>
  - DROP TRIGGER <name>
  - DROP INDEX <name>

```
DROP CONSTRAINT TRG_PERSON_INSERT

DROP TABLE PERSONS
```

Managing Access Permissions

- **GRANT command**

```
GRANT <persmission> ON <object> TO ROLE
```

  - **Example:**

```
GRANT SELECT ON PERSONS TO PUBLIC
```

- **REVOKE command**

```
REVOKE <persmission> ON <object> FROM ROLE
```

  - **Example:**

```
REVOKE SELECT ON PERSONS FROM HR
```

Creating Tables in MySQL

Defining Sequences, Triggers, Constraints

Creating Tables

Creating new table:

- **Define the table name**
- **Define the columns and their types**
- **Define the table primary key**
  - **Optionally you may mark this column as AUTO_INCREMENT so that you don't have to manually increment it.**
- **Define foreign/keys and constraints**

Creating Tables – Example

```
CREATE TABLE PERSONS (
 PERSON_ID INT UNSIGNED NOT NULL AUTO_INCREMENT,
 PRIMARY KEY(PERSON_ID),
 NAME VARCHAR(100) NOT NULL,
 DATE_OF_BIRTH DATE NOT NULL
);
```

SQL Language

Inserting Data in the Tables

Inserting Data

- `INSERT` command
  - `INSERT INTO <table> VALUES (<values>)`
  - `INSERT INTO <table>(<columns>) VALUES (<values>)`
  - `INSERT INTO <table> SELECT <values>`

```
INSERT INTO COUNTRY
VALUES ('1', 'Bulgaria', 'Sofia')

INSERT INTO COUNTRY(NAME, CAPITAL)
VALUES ('Bulgaria', 'Sofia')

INSERT INTO COUNTRY(COUNTRY_ID, NAME, CAPITAL)
SELECT NULL, COUNTRY, CAPITAL FROM CAPITALS
```

Inserting Selected Data

- We can insert into a table data selected from other SQL query
  - The columns an their type should match

```
INSERT INTO REGIONS(REGION_ID, REGION_NAME)
  SELECT 1000+EMPLOYEE_ID,
    LAST_NAME || '''s region'
  FROM employees

SELECT * FROM REGIONS
```

REGION_ID	REGION_NAME
1100	King's region
1101	Kochhar's region
…	…

SQL Language

Updating Data in the Tables

Updating Data

- `UPDATE` command
  - `UPDATE <table> SET <column=expression> WHERE <condition>`
  - Note: Don't forget the WHERE clause!

```
UPDATE PERSONS
SET NAME = 'Updated Name'
WHERE PERSON_ID = 1

UPDATE employees
SET SALARY = SALARY * 1.10
WHERE DEPARTMENT_ID = 3
```

Updating Joined Tables

- Updating joined tables is done by nested **SELECT**

```
UPDATE
   (SELECT SALARY
    FROM employees E INNER JOIN departments D
    ON E.DEPARTMENT_ID = D.DEPARTMENT_ID
    WHERE D.NAME = 'Accounting')
SET SALARY = SALARY * 1.10
```

SQL Language

Deleting Data from the Tables

Deleting Data

- Deleting rows from a table
  - `DELETE FROM <table> WHERE <condition>`

```
DELETE FROM PERSONS WHERE PERSON_ID = 1

DELETE FROM PERSONS WHERE NAME LIKE 'S%'
```

  - Note: Don't forget the `WHERE` clause!
- Delete all rows from a table at once
  - `TRUNCATE TABLE <table>`

```
TRUNCATE TABLE PERSONS
```

Delete From Joined Tables

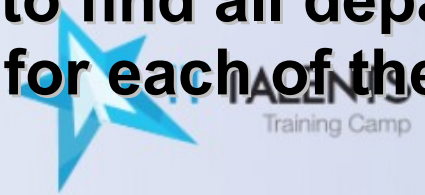
- Deleting from joined tables is done by nested `SELECT`
- Example: fire the entire IT staff

```
DELETE FROM employees
WHERE EMPLOYEE_ID IN
   (SELECT E.EMPLOYEE_ID
    FROM employees E JOIN departments D
      ON (E.DEPARTMENT_ID=D.DEPARTMENT_ID)
    WHERE D.DEPARTMENT_NAME='IT')
```

Questions?

Problems (1)

1. Write a SQL query to find the average salary in the "Sales" department. Use `AVG()`.
2. Write a SQL query to find the number of employees in the "Sales" department. Use `COUNT(*)`.
3. Write a SQL query to find the number of all locations where the company has an office.
4. Write a SQL query to find the number of all departments that has manager.
5. Write a SQL query to find the number of all departments that has no manager.
6. Write a SQL query to find all departments' names and the average salary for each of them.



Problems (2)

7. Write a SQL query to find the count of all employees in each department. Display the name, location and number of employees for each department.
8. Write a SQL query to find for each department and for each manager the count of all corresponding employees.
9. Write a SQL query to find all managers that have exactly 5 employees. Display their names and the name and location of their department.
10. Write a SQL query to find the total number of employees for each region.
11. Write a SQL query to find for each department and for each job title the total number of employees.

Problems (3)

12. Write a SQL query to find the names and salaries of the employees that take the minimal salary in the company. Use nested `SELECT` statement.
13. Write a SQL query to find the names and salaries of the employees that get a salary that is up to 10% higher than the minimal salary for the company.
14. Write a SQL that displays all departments and the highest salary for each department along with the name of the employee that takes it. If multiple employees in the same department have highest salary, display the first of them.
15. Write a SQL query to find the names of all employees whose last name is exactly 5 characters long.

Problems (4)

16. Write a SQL query to find the names of all employees whose first name and last name start with the same letter.
17. Display all departments names and their manager's name. For departments without manager display "(No manager)".
18. Display all employees along with their number of directly managed people. For employees not managing anybody display "Just and employee". For employees managing only 1 employee display "Junior manager". .
19. Write a SQL query to print the current date and time in the format " hour:minutes:seconds day-month-year". Display also the date coming after a week.

Problems (5)

20. Write a SQL statement to create a table `USERS`. Users should have username, password, full name and last login time. Choose appropriate data types for the fields of the table. Define a primary key column with a primary key constraint. Define a trigger to automatically fill the full name column value before inserting a record.
21. Write a SQL statement to create a view that displays the users from the `USERS` table that have been in the system today. Test if the view works correctly.
22. Write a SQL statement to create a table `GROUPS`. Groups should have unique name (use unique constraint). Define primary key and a trigger for populating it.

Problems (6)

23. Write a SQL statement to add a column `GROUP_ID` to the table `USERS`. Fill some data in this new column and as well in the GROUPS table. Write a SQL statement to add a foreign key constraint between tables `USERS` and `GROUPS`.
24. Write SQL statements to insert several records in the `USERS` and `GROUPS` tables.
25. Write SQL statements to insert in the `USERS` table the names of all employees from the `employees` table. Combine the first and last names as a full name. For user name use the email column from `employees`. Use blank password.
26. Run the above 10 times to generate enough testing data for the USERS table.

Problems (7)

27. Write a SQL statement that changes the password to `NULL` for all `USERS` that have not been in the system since 10.03.2006. Select table data to see the changes.
28. Write a SQL statement that deletes all users without passwords (`NULL` or empty password). Select table data to see the changes.
29. Write a SQL query to list all users whose username starts with `'s'` and the number of groups for each of them.

Problems (8)

31. Define table WORKHOURS to store work reports for each employee (date, task, hours, comments).
32. Define foreign key between the tables WORKHOURS and EMPLOYEE. Add additional column in the employee table if needed.
33. Write several SQL statements to fill some data in the WORKHOURS table.
34. Write a SQL query to find all the average work hours per week for each country.
35. Write a SQL query to find all the departments where some employee worked overtime (over 8 hours/day) during the last week.

Problems (9)

36. Write a SQL query to find all employees that have worked 3 or more days overtime in the last week. Display their name, location department and country.