Lection 4

SQL
History, basics

History

System/R

SEQUEL (Structured English Query Language), 1974

SEQUEL/2, 1976-1977

SQL89 → SQL1

SQL92 → SQL2

SQL99 → SQL3

SQL:2003

SQL Types

Interactive

Internal

Subsets SQL

DDL

DML

TCL

DCL

CCL

DDL

Data Definition Language manages table, index structure, objects.

CREATE/DROP/ALTER TABLE
CREATE/DROP VIEW
CREATE/DROP INDEX
CREATE/DROP SCHEME
CREATE/DROP/ALTER DOMAIN

DML

Data Manipulation Language is the subset of SQL used to add, update and delete data.

SELECT
INSERT
DELETE
UPDATE

TCL

Transaction Control Language are used to control changes that are made by DML commands.

COMMIT
ROLLBACK
SET TRANSACTION

DCL

Data Control Language, another name is Access Control Language, are used for administrative functions, to give or to take rights/access to use database, data or to execute commands.

GRANT
REVOKE

CCL

Cursor Control Language are used to define cursors, prepare SQL commands.

DECLARE/OPEN/CLOSE CURSOR
EXPLAIN
FETCH
PREPARE
EXECUTE
DESCRIBE

To mention

- Rows are not sorted, order of data is absent;
- Columns are named and have numbers / order;
- Datatypes: INTEGER/DECIMAL, CHAR/VARCHAR;
- Datetypes: DATA/DATE/TIME/DATETIME/MONEY/BINARY и т.д.;
- USER;
- End of query «;»;
- Text and values «'»/«"».

Definitions

Keywords: SELECT, INSERT

Variables: sname, city, a > b

Required(<>): <field>

Optional([]): [WHERE <condition>]

Complex content ({}): {field definition}

Choice, one from a list (|): *|<field>

Create a table

CREATE TABLE <tablename>(
{field definition}
[, {field definition}]
[,...]
[,{field constraints}]
[,...]
);

field definition
<field name> <datatype>[(size)] [field constraints] [DEFAULT <value>]

Create a table

Field constraints
NOT NULL | NULL — field cannot/can has NULL-values
PRIMARY KEY — field is a primary key
REFERENCES <table name> (<field name>) - field is a foreign key for specified field in another table

Table constrains
PRIMARY KEY (<field name>[,<field name>][,...])
FOREING KEY (<field name>[,<field name>][,...]) REFERENCES <table name> (<field name>[,<field name>][,...])

Add data

INSERT INTO <table name>[(<field name>[,...])]
VALUES(<value>[,...]);

Getting data

```
SELECT field1, field2 FROM table1;
SELECT field2, field1 FROM table1;

SELECT name, salary*2 FROM persons;

SELECT 'FIO',name,'SALARY:',salary FROM persons;
```

Delete data

DELETE FROM <table name> [WHERE <condition>];

Getting data

SELECT *|<field name>[,<field name> …]
FROM <table name>
[WHERE <condition>]
[ORDER BY <field name>|<field number>[,...]];

SELECT fio FROM students;

SELECT * FROM students WHERE age < 18;

Conditions

WHERE {condition} [<boolean operator> {condition}] [...]

[<argument1>] <operator> <argument2>

Relational operators

=

>

<

>=

<=

<>

Boolean operators

<argument1> AND <argument2>

<argument1> OR <argument2>

NOT <argument1>

IN, BETWEEN

Operator **IN** defines set of values to check field value.

<field name> IN (<value>[,...])

Operator BETWEEN looks like IN. The difference in values. IN defins set of values but BETWEEN defines range of values.

<field name> BETWEEN <minimum> AND <maximum>

LIKE

LIKE can be use only with CHAR or VARCHAR to find substrings. In condition we can use wildcards — special symbols.

<field name> LIKE <substring>

There are two symbols for wildcards in LIKE:
• Under stroke (_) to replace one character. For example, 'b_t' will be equal to 'bat' or 'bit' but will not equal 'brat'.
• Percent (%) replaces sequence of any length of characters. Example, '%p%t' will be equal to 'put', 'posit', or 'opt' but no 'spite'.

NULL-values

IS [NOT] NULL

SELECT * FROM students WHERE age IS NULL

SELECT * FROM students WHERE age IS NOT NULL

SELECT * FROM students WHERE age NOT IN (18,19)

SELECT * FROM students WHERE NOT age IN (18,19)

Aggregate functions

COUNT(*|[DISTINCT|ALL] <field name>)

SUM([DISTINCT|ALL] <field name>|<expression>)

AVG([DISTINCT|ALL] <field name>|<expression>)

MAX([DISTINCT|ALL] <field name>|<expression>)

MIN([DISTINCT|ALL] <field name>|<expression>)

Aggregate functions

```
SELECT COUNT(*) FROM students;

SELECT COUNT(DISTINCT fio) FROM students;

SELECT MAX(age) FROM students;

SELECT MIN(age) FROM students;

SELECT AVG(age) FROM students;

SELECT MIN(price*0.5) FROM goods;
```

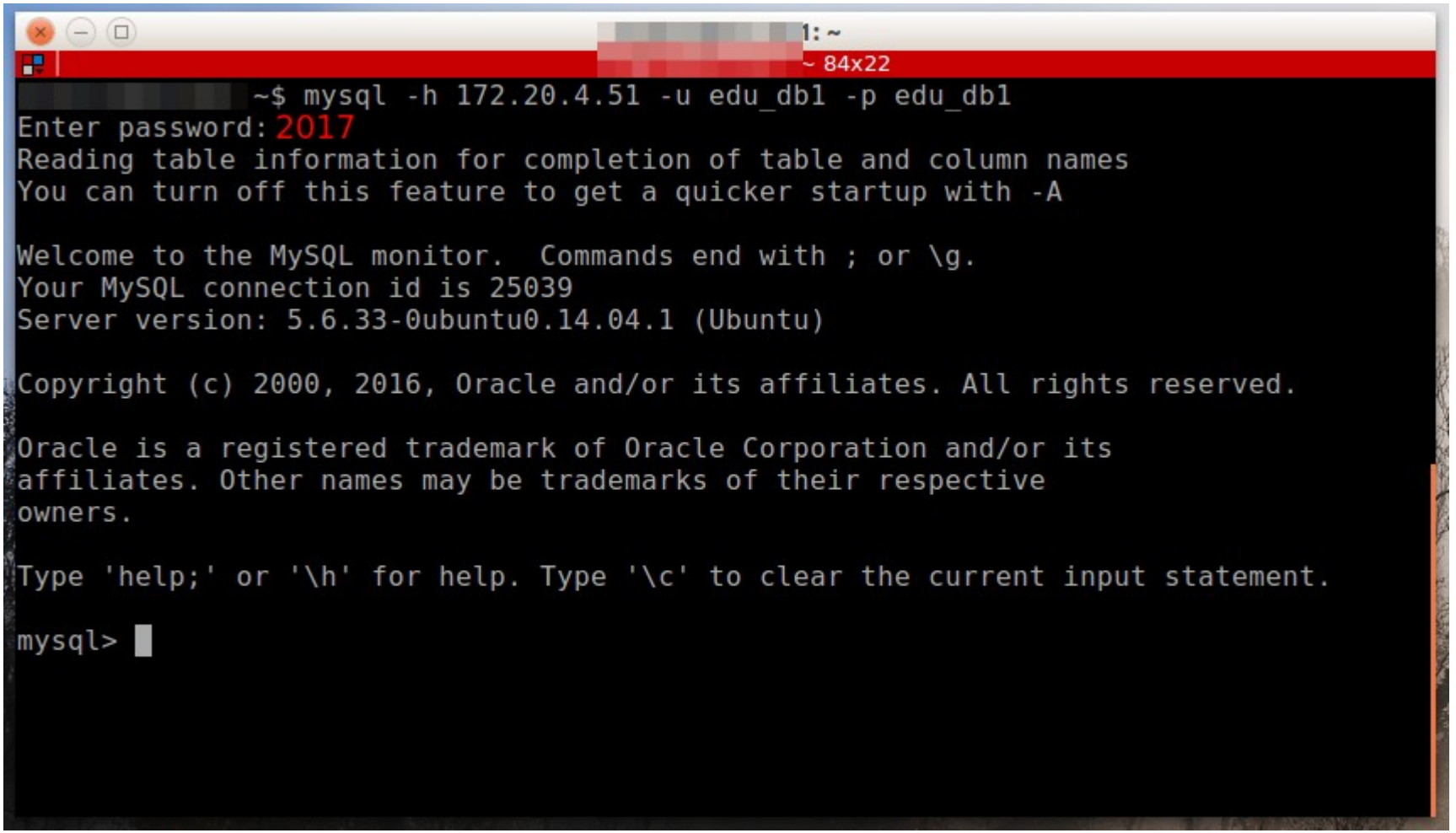
Group

GROUP BY <field name>
HAVING <condition>

SELECT {set of fields}, <aggregate function>[, ...]
FROM <table name>
GROUP BY {set of fields}
[HAVING {condition}]

Additional info

- press Ctrl+Alt+T in Linux



Task 1

1) Make a new table **edu_test** with three fields: tnum (for a test number), tdate (for a event date), tres (for a test result)
2) Use command **SHOW TABLES** to see list of tables
3) Insert a record into **edu_test**
4) See records from **edu_test**
5) Disconnect from databse
6) Connect to database
7) See records from **edu_test**
8) Destroy this table **edu_test**
9) See records from **edu_test**
10) Use command **SHOW TABLES** to see list of tables

Task 2

1) Put a record with information about a saler: city - San Jose, name — Bianco, comm — NULL, snum — 1100
2) Delete all Clemens orders
3) Increase rating for all customers from Rome by 100
4) Get all orders with amount more than $1,000
5) Get fields sname and city for all salers from London with commission more than .10
6) Summarize all amount for 03/10/2016
7) Calculate different customer's cities (do not calculate underfined)

Task 3

1) Get orders with minimal amount for every customers
2) Get all customers with name which starts with letter **G**
3) Find out maximal rating in every city
4) Get orders with number of order, saler's number, commission for the saler (let's pretend it's 12%)
5) Find highest rating in every city. Output must be like

*For the city **city** the highest rating is: **rating***

6) Get list of customers in descendant order for names