

# MySQL Basics II

**“Code with Passion!”**



# Topics

- Advanced field modifiers
  - > AUTO\_INCREMENT
  - > INDEX
  - > UNIQUE
- Table modifiers
  - > Storage Engine
  - > Other modifiers
- WHERE clause options
- GROUP BY and HAVING
- User-defined variables

# Advanced Field Modifiers

# Advanced Field Modifiers

- AUTO\_INCREMENT
  - > MySQL automatically generates a number (by incrementing the previous value by 1)
  - > Used for creating primary key automatically
- INDEX
  - > Index a field
  - > When a field is indexed, MySQL no longer must scan the whole table, instead uses the index to locate the record(s)
  - > Performance booster
- UNIQUE
  - > The value must be unique

# AUTO\_INCREMENT

```
/* Create "employees" table */
DROP TABLE IF EXISTS employees;

CREATE TABLE employees (
    /* If value of this field is not provided, one will be created by MySQL */
    employee_id int(11) NOT NULL AUTO_INCREMENT,
    name varchar(255) NOT NULL,
    salary decimal(7,2) NOT NULL,
    PRIMARY KEY (employee_id)
);
```

```
/* Data for the table employees - providing employee_id explicitly */
INSERT INTO employees(employee_id, name, salary)
VALUES
(1,'jack','3000.00'),
(2,'mary','2500.00'),
(3,'nichole','4000.00');
```

```
/* Data for the table employees - using AUTO_INCREMENT */
INSERT INTO employees(name, salary)
VALUES
('angie','5000.00'),
('jones','5000.00');
```

# INDEX constraint

```
CREATE TABLE employees (
    employee_id int(11) NOT NULL AUTO_INCREMENT,
    name varchar(255) NOT NULL UNIQUE,
    department varchar(255) NOT NULL,
    salary decimal(7,2) NOT NULL,
    PRIMARY KEY (employee_id),
    INDEX (department)
);
```

# UNIQUE constraint

```
CREATE TABLE employees (
    employee_id int(11) NOT NULL AUTO_INCREMENT,
    /* name field now has UNIQUE constraint */
    /* every name in this field must be unique */
    name varchar(255) NOT NULL UNIQUE,
    salary decimal(7,2) NOT NULL,
    PRIMARY KEY (employee_id)
);
```

```
/* Data for the table employees - using AUTO_INCREMENT */
INSERT INTO employees(name, salary)
VALUES
('angie','5500.00'),
('jones','5000.00'),
('jones','4000.00'); /* This should result in an error */
```

# Lab:

**Exercise 1: Field Modifiers**  
**1611\_mysql\_basics2.zip**



# Table Modifiers: **Storage Engine**

# What is a Storage Engine?

- A "storage engine" is the underlying software component that a database management system (DBMS) uses for performing database operations
- Represents table type
  - > A table is associated with a particular storage engine
  - > A table is either created with a particular storage engine or altered to a different storage engine

# Storage Engines

- MySQL support a set of storage engines based on pluggable storage engine architecture
- Each storage engine has its own advantages and disadvantages
  - > Choosing a wrong one might cause performance drag
- Different storage engines can be assigned to different tables in a single database

# Factors to consider when choosing a Storage engine for a table

- Frequency of reads vs. writes
  - > *MyISAM* would be better choice if the table access is mostly reads (SELECT)
- Whether transactional support is needed or not
  - > Only *InnoDB* supports transactional behavior
- Indexing requirement
- OS portability
- Future extensibility and changeability

# Storage Engines in MySQL

- InnoDB
  - > InnoDB is a transaction-safe (ACID compliant) storage engine for MySQL that has commit, rollback, and crash-recovery capabilities to protect user data
  - > Default
- MyISAM
  - > It is based on the older ISAM code but has many useful extensions
- MRG\_MYISAM
  - > Is a collection of identical MyISAM tables that can be used as one.

# Storage Engines

- FEDERATED
  - > Enables data located on a remote MySQL database can be accessed through local server
- ARCHIVE
  - > Used for storing large amounts of data without indexes in a very small footprint.
- CSV
  - > Stores data in text files using comma-separated values format.
- BLACKHOLE
  - > Acts as a “black hole” that accepts data but throws it away and does not store it.

# Storage Engines

- MRG\_MYISAM
  - > Is a collection of identical MyISAM tables that can be used as one.
- MEMORY (HEAP)
  - > Hash based, stored in memory, useful for temporary tables

# Creating a table with ENGINE

```
mysql> CREATE TABLE t1_InnoDB (id int) ENGINE = InnoDB;  
Query OK, 0 rows affected (0.18 sec)
```

```
mysql> CREATE TABLE t2_MyISAM (id int) ENGINE = MyISAM;  
Query OK, 0 rows affected (0.07 sec)
```

```
// Create a table with a default storage engine  
mysql> CREATE TABLE t3_default (id int);  
Query OK, 0 rows affected (0.13 sec)
```

# **Table Modifiers: Misc. Modifiers**

# Table Modifiers

- AUTO\_INCREMENT
  - > Specifies the starting value of the AUTO\_INCREMENT field
- CHARACTER SET, COLLATE
  - > Specifies the table character set and collation
- CHECKSUM
  - > Specifies whether the table checksum should be computed and stored
- MAX\_ROWS, MIN\_ROWS
  - > Specifies the maximum and minimum number of rows a table can have

# Table Modifiers (Continued)

- **PACK\_KEYS**
  - > Specifies whether indexes should be compressed or not
- **DELAY\_KEY\_WRITE**
  - > Specifies whether indexes should be updated only after all writes to the table are complete
  - > Can improve performance for tables with high frequency of writes
- **DATA DIRECTORY**
  - > Specifies non-default data directory
- **INDEX DIRECTORY**
  - > Specifies non-default index directory

# Lab:

**Exercise 2: Table Modifiers**  
**1611\_mysql\_basics2.zip**



# **WHERE Clause Options**

# Comparison Operators in WHERE

- =,>,<,.>=,<=,<>

```
SELECT * FROM employees  
WHERE salary > 3500;
```

# Logical Operators in WHERE

- AND, OR, NOT

```
SELECT * FROM employees  
WHERE (department_id = 1 AND NOT name = 'nichole')  
OR salary > 4500;
```

# BETWEEN

```
SELECT * FROM employees  
WHERE salary BETWEEN 2000 AND 4000;
```

IN

```
SELECT * FROM employees  
WHERE name IN ('nichole', 'jack');
```

# LIKE

```
SELECT * FROM employees  
WHERE name LIKE '%n%';
```

```
SELECT * FROM employees  
WHERE name LIKE '%e';
```

# Regular Expression

```
/* Get all records whose name is either 'jones' or 'mary' */
```

```
SELECT * FROM employees
```

```
WHERE name REGEXP 'jones|mary';
```

```
/* Get all records whose name starts with 'j' */
```

```
SELECT * FROM employees
```

```
WHERE name REGEXP '^j';
```

```
/* Get all records whose name ends with 'e' */
```

```
SELECT * FROM employees
```

```
WHERE name REGEXP 'e$';
```

# DISTINCT

SELECT DISTINCT salary FROM employees;

# Lab:

**Exercise 3: Where Clause**  
**1611\_mysql\_basics2.zip**



# **GROUP BY and HAVING**

# GROUP BY

- Returns group of rows
- Divides a table into sets and it is usually used with SQL aggregate functions, like `COUNT(..)`, which produces summary value for each set

# GROUP BY Example

```
/* Data for the table employees */
```

```
INSERT INTO employees(name, salary, department_id) VALUES  
('jack','3000.00', 1),  
('mary','2500.00', 2),  
('nichole','4000.00', 1),  
('angie','5000.00', 2),  
('jones','5000.00', 3);
```

```
/* Get number of employees for each department using GROUP BY */
```

```
SELECT department_id, COUNT(employee_id) AS employee_count  
FROM employees  
GROUP BY department_id;
```

department_id	employee_count
1	2
2	2
3	1

# HAVING

- HAVING clause is like a WHERE clause for groups
  - > Just as WHERE clause limits rows, HAVING clause limits groups
- In most programming contexts, you will use HAVING clause after GROUP BY clause to limit groups by searched conditions

# HAVING Example

```
/* Data for the table employees */
INSERT INTO employees(name, salary, department_id) VALUES
('jack','3000.00', 1),
('mary','2500.00', 2),
('nichole','4000.00', 1),
('angie','5000.00', 2),
('jones','5000.00', 3);

/* Get number of employees for each department using GROUP BY &
 * the number of employees are greater than or equal to 2. */
SELECT department_id, COUNT(employee_id) AS employee_count
FROM employees
GROUP BY department_id
HAVING employee_count >= 2;
```

department_id	employee_count
1	2
2	2

# Lab:

**Exercise 4: GROUP BY & HAVING**  
**1611\_mysql\_basics2.zip**



# User-defined Variables

# What are User-defined variables?

- You can store a value in a user-defined variable in one statement and then refer to it later in another statement
  - > This enables you to pass values from one statement to another
- User-defined variables are connection-specific
  - > A user variable defined by one client cannot be seen or used by other clients
  - > All variables for a given client connection are automatically freed when that client exits

# How to create user-defined variables

- User variables are written as `@var_name`
- One way to set a user-defined variable is by issuing a SET statement:
  - > `SET @var_name = expr [, @var_name = expr] ...`
  - > For SET, either `=` or `:=` can be used as the assignment operator
- Another way to define a user-defined variable is by using `SELECT.. INTO`
  - > `SELECT .. INTO @var_name`

# Setting User-defined Variables with SET

```
mysql> SET @my_var1 = 10, @my_var2 := 20;  
Query OK, 0 rows affected (0.24 sec)
```

```
mysql> SELECT @my_var1, @my_var2, @my_var3 := @my_var1 + @my_var2;  
+-----+-----+  
| @my_var1 | @my_var2 | @my_var3 := @my_var1 + @my_var2 |  
+-----+-----+  
|      10 |      20 |                  30 |  
+-----+-----+  
1 row in set (0.05 sec)
```

```
mysql> SET @my_string_var = 'Sang Shin';  
Query OK, 0 rows affected (0.00 sec)
```

```
mysql> SELECT UPPER(@my_string_var), LOWER(@my_string_var);  
+-----+-----+  
| UPPER(@my_string_var) | LOWER(@my_string_var) |  
+-----+-----+  
| SANG SHIN           | sang shin        |  
+-----+-----+  
1 row in set (0.08 sec)
```

# Setting User-defined Variables with SELECT

```
mysql> SELECT 67 INTO @my_var4;
```

```
Query OK, 1 row affected (0.06 sec)
```

```
mysql> SELECT @my_var4;
```

```
+-----+  
| @my_var4 |  
+-----+  
|    67 |  
+-----+  
1 row in set (0.00 sec)
```

```
mysql> SELECT @my_var4 + 10 INTO @my_var5;
```

```
Query OK, 1 row affected (0.00 sec)
```

```
mysql> SELECT @my_var4, @my_var5;
```

```
+-----+-----+  
| @my_var4 | @my_var5 |  
+-----+-----+  
|    67 |     77 |  
+-----+-----+  
1 row in set (0.00 sec)
```

# Setting User-defined Variables with SELECT

```
mysql> select name into @first from employees where employee_id =1;  
Query OK, 1 row affected (0.00 sec)
```

```
mysql> select @first;  
+-----+  
| @first |  
+-----+  
| jack |  
+-----+  
1 row in set (0.00 sec)
```

```
mysql> select sum(salary) from employees into @total;  
Query OK, 1 row affected (0.00 sec)
```

```
mysql> select @total;  
+-----+  
| @total |  
+-----+  
| 19500.00 |  
+-----+  
1 row in set (0.00 sec)
```

# Lab:

**Exercise 5: User Defined Variables**  
**1611\_mysql\_basics2.zip**



# Code with Passion!

