2018

CI6206 Internet Programming

MySQL

Wong Twee Wee

Ver1.1

TOPICS

- Introduction to MySQL
- Connecting and Disconnecting
- Entering Basic Queries
- Creating and Using a Database

ATTRIBUTION

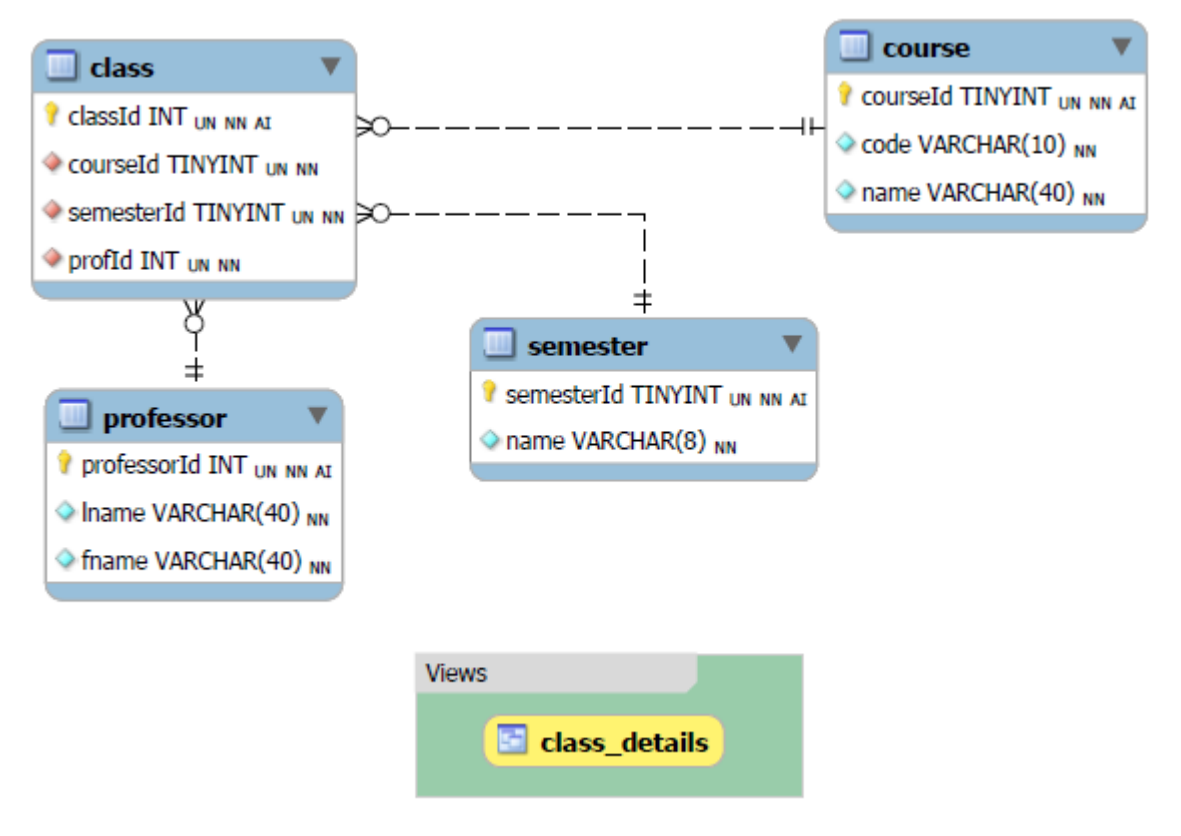
- Most of these slides are based directly on the MySQL Documentation.
- Most information comes from Chapter 3, MySQL Tutorial:
- *http://dev.mysql.com/doc/refman/5.6/en/index.html*
- *http://dev.mysql.com/doc/refman/5.6/en/tutorial.html*

ER DIAGRAM

The main value of carefully constructing an ERD is that it can readily be converted into a **database structure**.

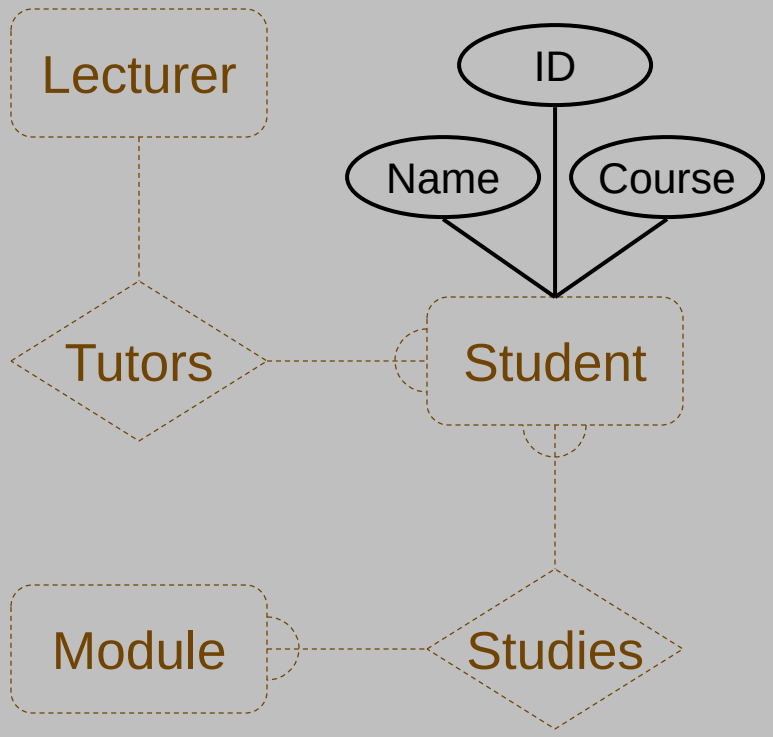
There are **three components** in ERD.

- Entities: Number of tables you need for your database.
- Attributes: Information such as property, facts you need to describe each table.
- Relationships: How tables are linked together.



DIAGRAMMING ATTRIBUTES

- In an E/R Diagram attributes may be drawn as ovals
- Each attribute is linked to its entity by a line
- The name of the attribute is written in the oval



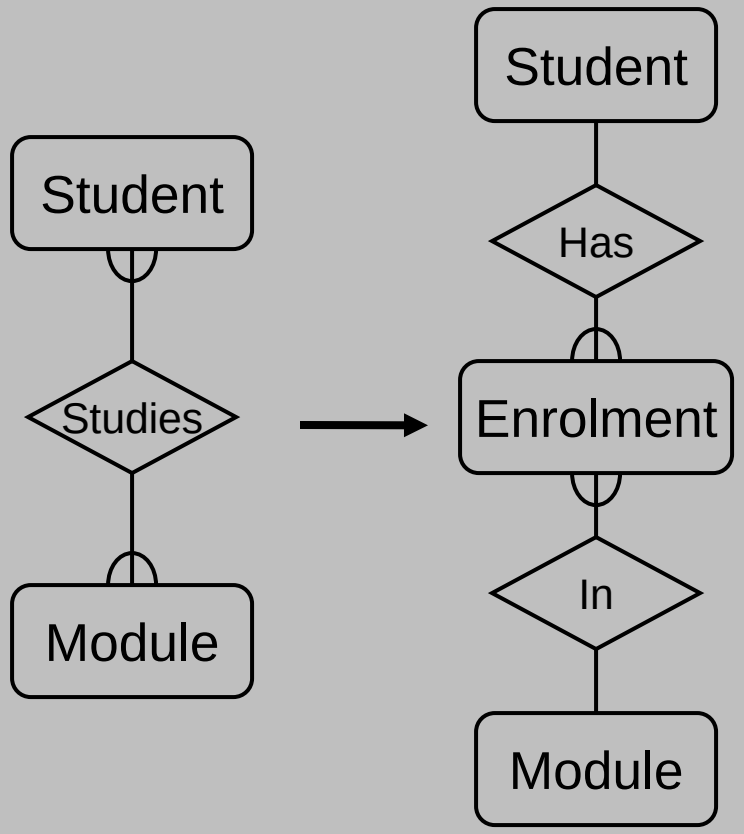
CARDINALITY RATIOS

- Each entity in a relationship can participate in zero, one, or more than one instances of that relationship
- This leads to 3 types of relationship…

- One to one (1:1)
  - Each lecturer has a unique office
- One to many (1:M)
  - A lecturer may tutor many students, but each student has just one tutor
- ~~Many to many (M:M)~~
  - Each student takes several modules, and each module is taken by several students

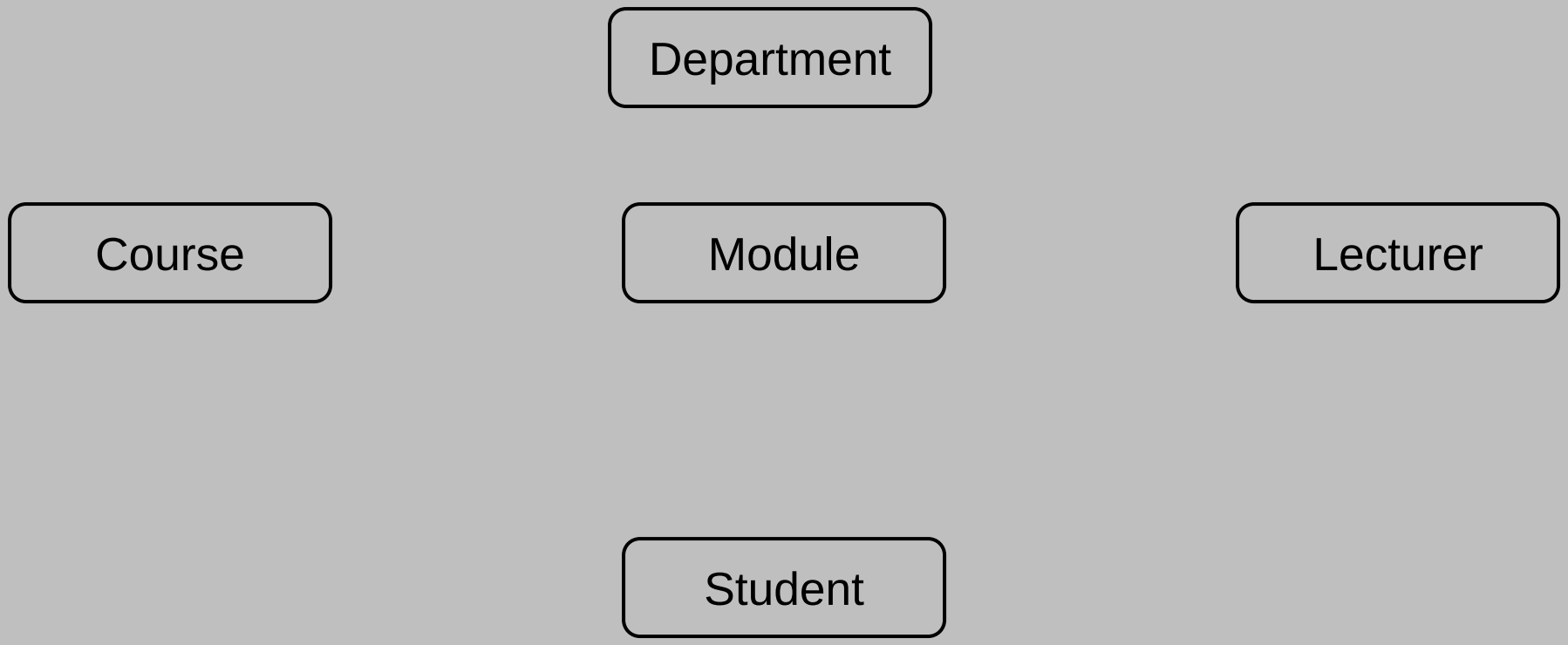
REMOVING M:M RELATIONSHIPS

- Many to many relationships are difficult to represent
- We can split a many to many relationship into two one to many relationships
- An entity represents the M:M relationship



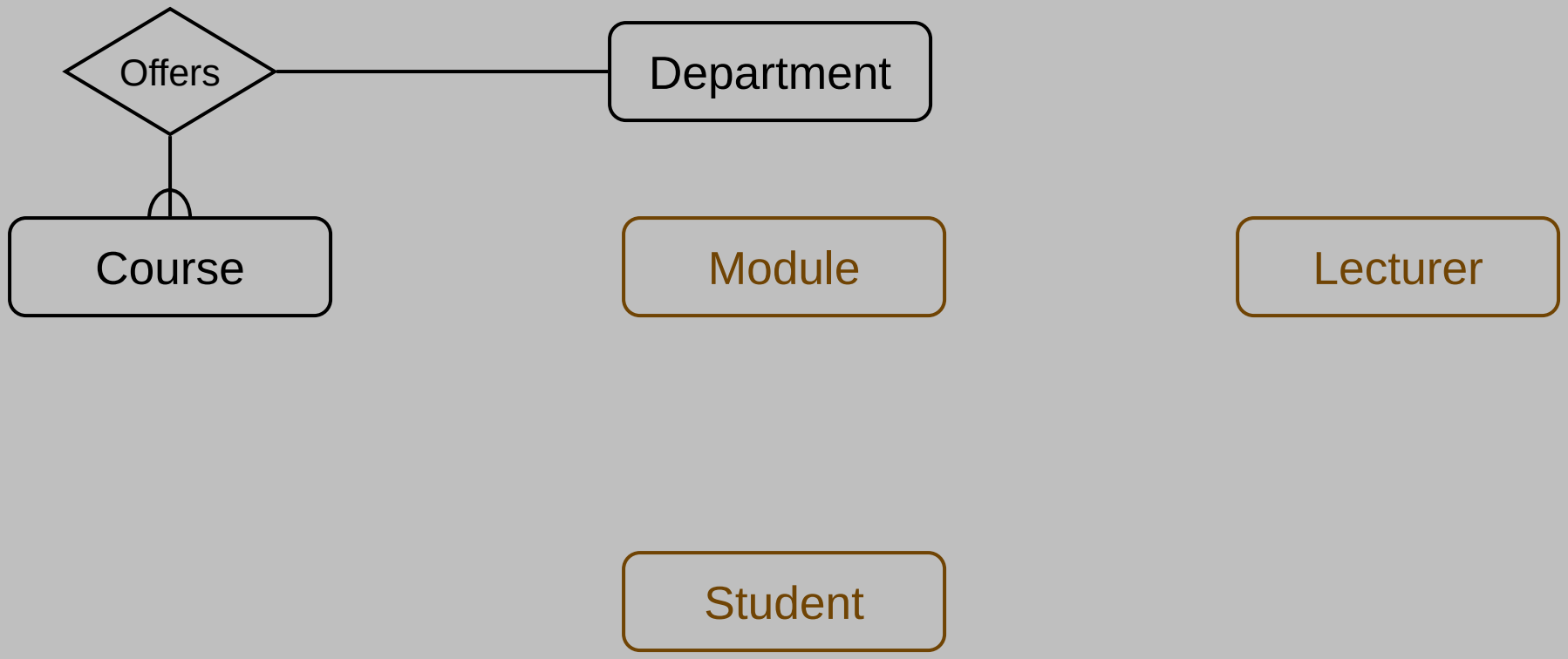
EXAMPLE - E/R DIAGRAM

Entities: Department, Course, Module, Lecturer, Student



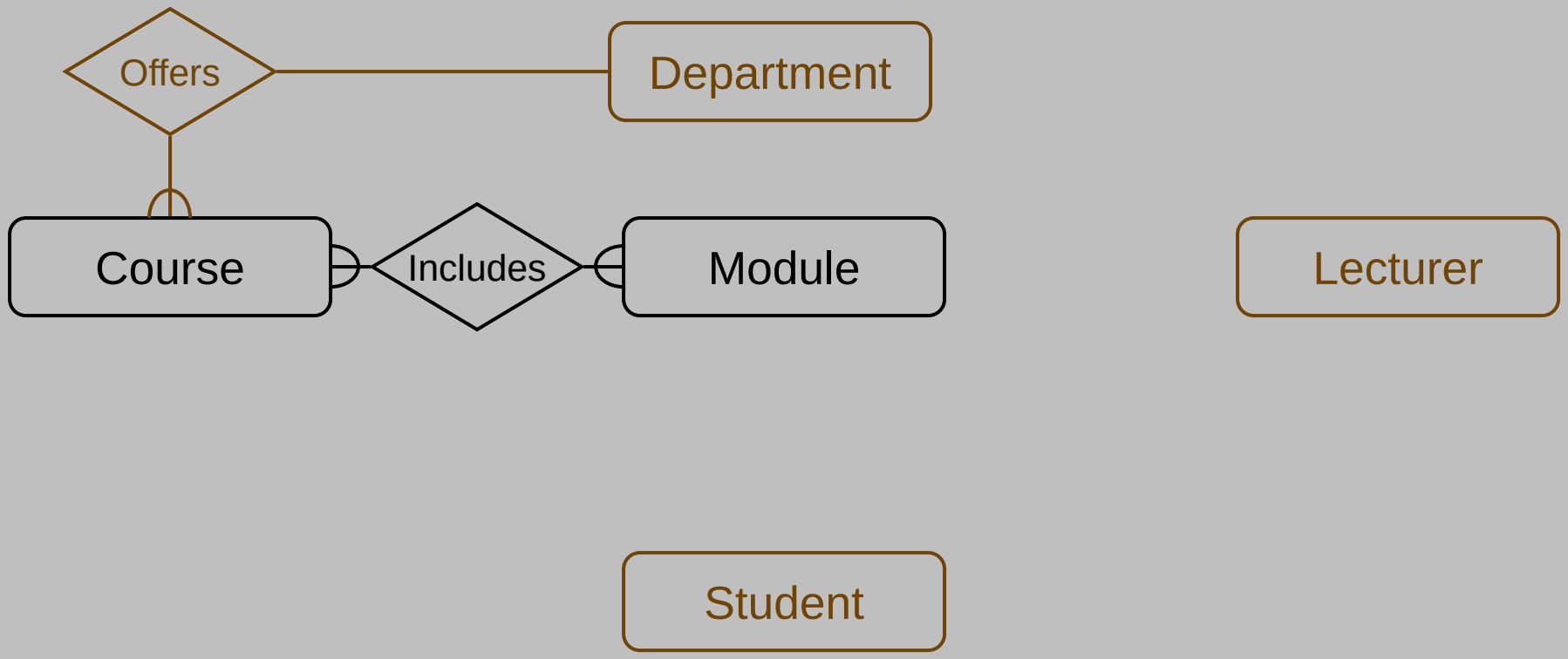
EXAMPLE - E/R DIAGRAM

Each department offers several courses



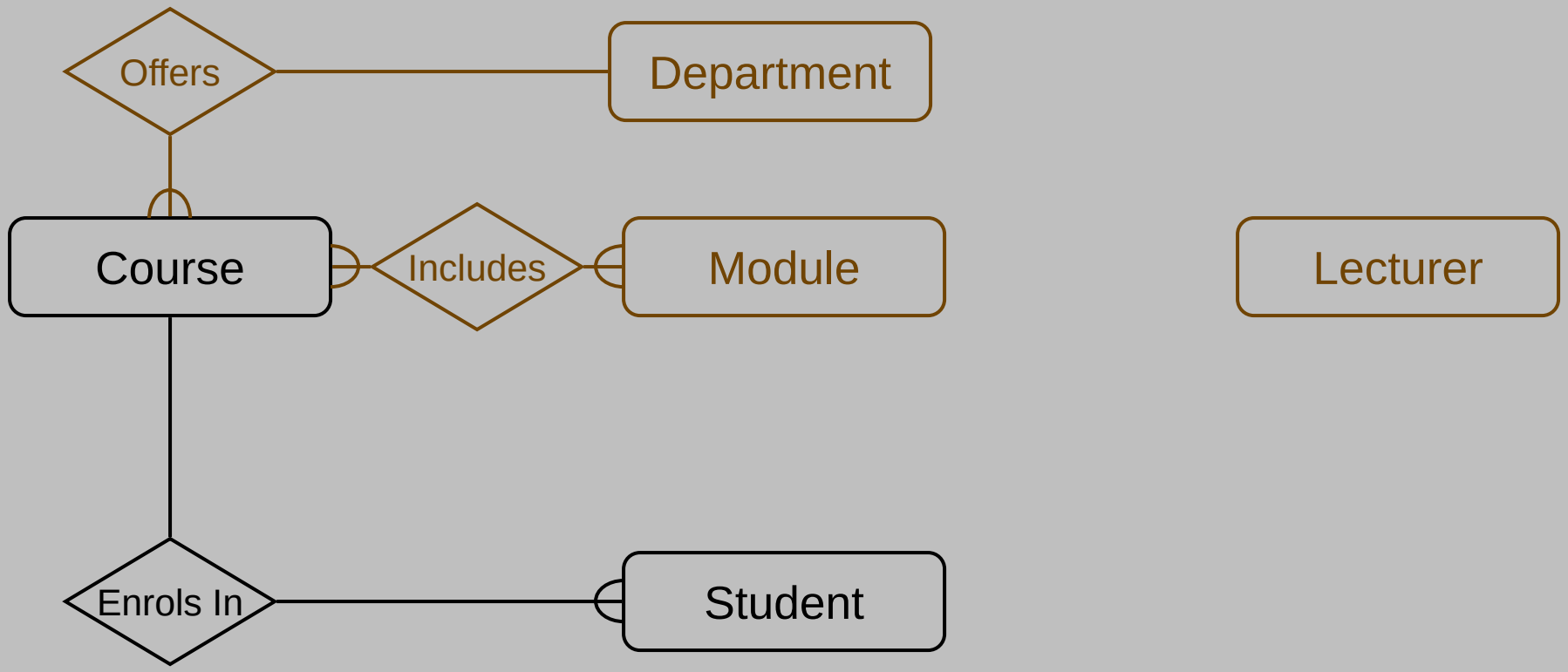
EXAMPLE - E/R DIAGRAM

A number of modules make up each courses



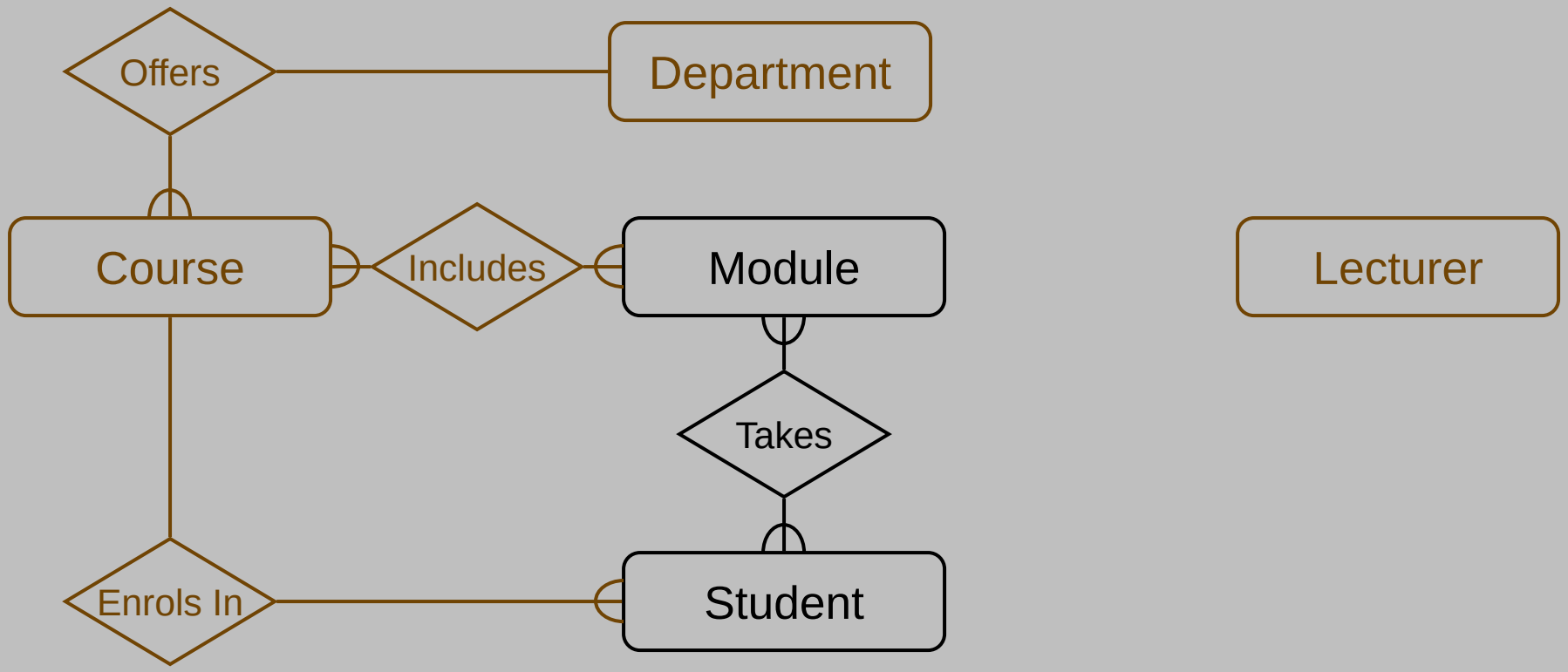
EXAMPLE - E/R DIAGRAM

Students enrol in a particular course



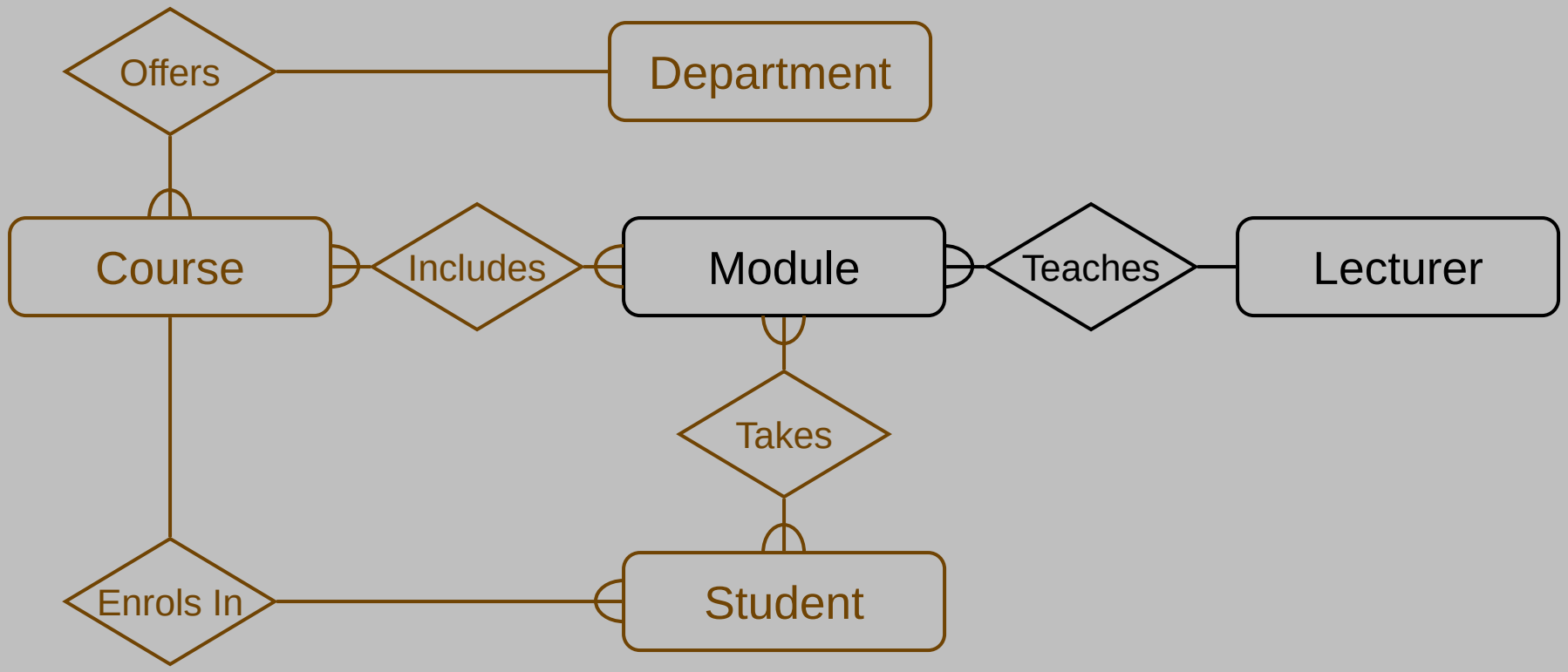
EXAMPLE - E/R DIAGRAM

Students … take modules



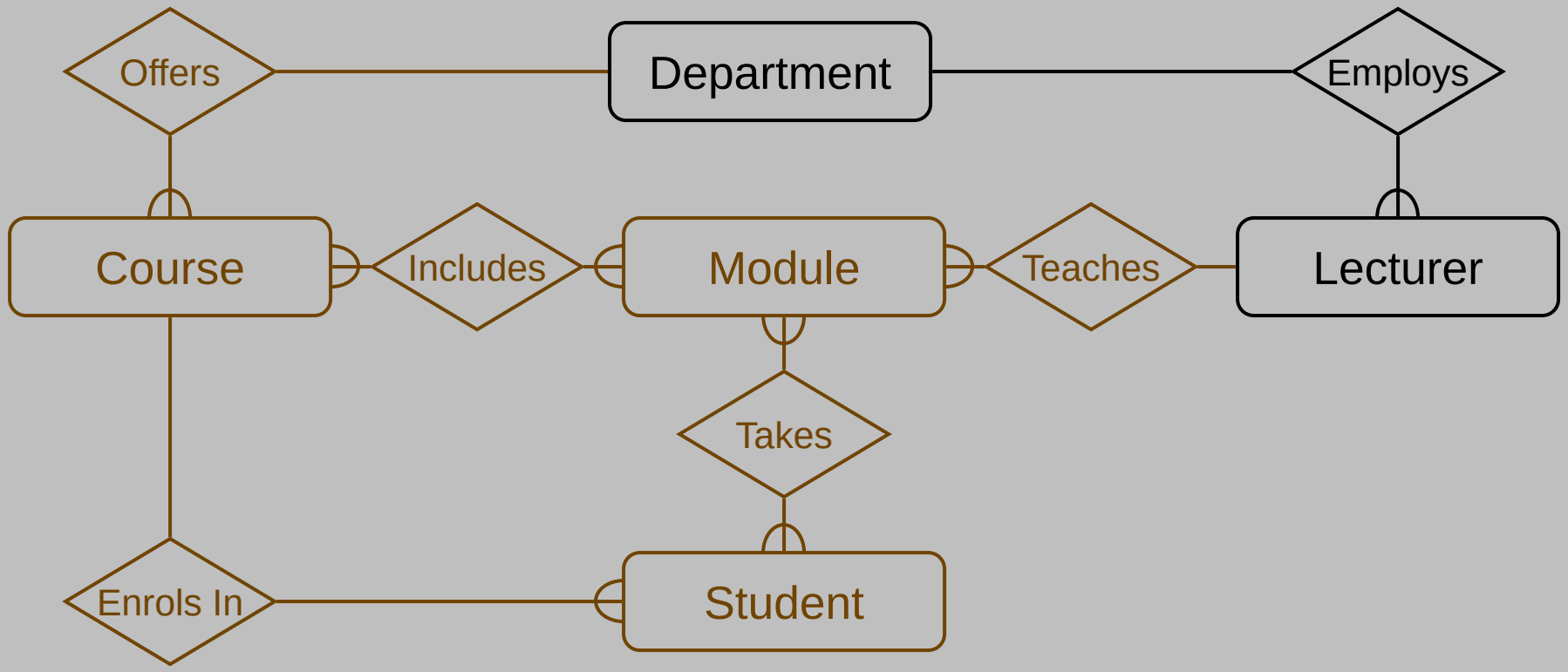
EXAMPLE - E/R DIAGRAM

Each module is taught by a lecturer



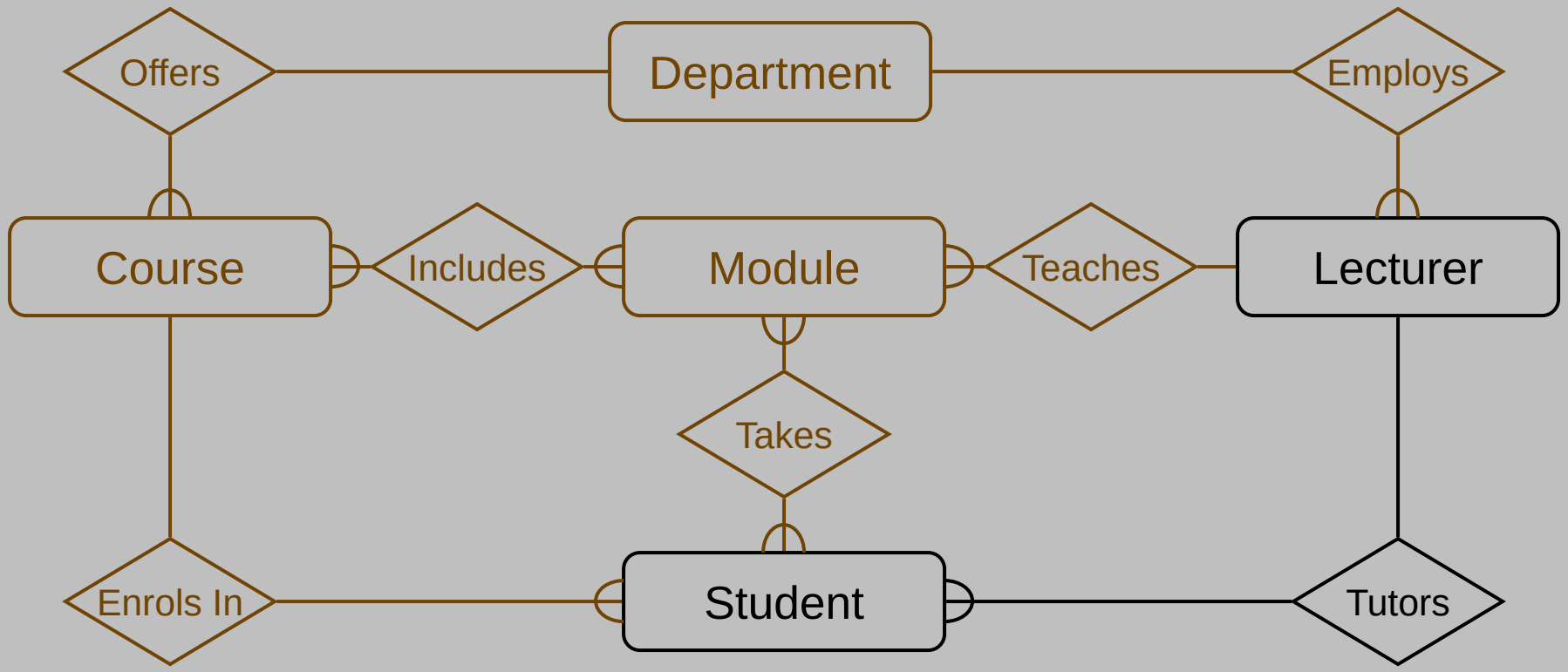
EXAMPLE - E/R DIAGRAM

a lecturer from the appropriate department



EXAMPLE - E/R DIAGRAM

each lecturer tutors a group of students



MYSQL

- MySQL is a very popular, open source database.
- Handles very large databases; very fast performance.
- Why are we using MySQL?
  - Free (much cheaper than Oracle!)
  - Each student can install MySQL locally.
  - Easy to use Shell for creating tables, querying tables, etc.

Need JDBC to connect to MySQL.

CRASH COURSE FUNDAMENTALS

- In order to use JDBC, you need:
  - a database.
  - basic understand of SQL (Structured Query Language)
- Some students may have database backgrounds; others may not.
- The purpose of this lecture is to get all students up to speed on database fundamentals.

CONNECTING TO AND USING MYSQL

- Method 1
  - Interactive shell for creating tables, inserting data, etc.
  - On Windows, just go to c:\mysql\bin, and type:
  - `mysql`
- Method 2
  - GUI – MySQL Workbench 6.3 and above.
  - http://www.mysql.com/products/workbench/

SAMPLE SESSION

- **For example:**

```
Enter password:  *****
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 241 to server version: 3.23.49

Type 'help;' or '\h' for help. Type '\c' to clear the buffer.

mysql>
```

- **To exit the MySQL Shell, just type QUIT or EXIT:**

```
mysql> QUIT
mysql> exit
```

BASIC QUERIES

- Once logged in, you can try some simple queries.
- For example:

```
mysql> SELECT VERSION(), CURRENT_DATE;
+------------+--------------+
| VERSION()  | CURRENT_DATE |
+------------+--------------+
| 3.23.49    | 2002-05-26   |
+------------+--------------+
1 row in set (0.00 sec)
```

- Note that most MySQL commands end with a semicolon (;)
- MySQL returns the total number of rows found, and the total time to execute the query.

BASIC QUERIES

- Keywords may be entered in any lettercase.
- The following queries are equivalent:

NOT CASE SenSiTive

```
mysql> SELECT VERSION(), CURRENT_DATE;
mysql> select version(), current_date;
mysql> SeLeCt vErSiOn(), current_DATE;
```

BASIC QUERIES

- You can also enter multiple statements on a single line. Just end each one with a semicolon:

```
mysql> SELECT VERSION(); SELECT NOW();
+---------------+
| VERSION()     |
+---------------+
| 3.22.20a-log |
+---------------+
+----------------------+
| NOW()                |
+----------------------+
| 2004 00:15:33 |
+----------------------+
```

MULTI-LINE COMMANDS

- mysql determines where your statement ends by looking for the terminating semicolon, not by looking for the end of the input line.
- Here's a simple multiple-line statement:

```
mysql> SELECT
    -> USER()
    -> ,
    -> CURRENT_DATE;
+---------------------+---------------+
| USER()              | CURRENT_DATE  |
+---------------------+---------------+
| joesmith@localhost  | 1999-03-18    |
+---------------------+---------------+
```

CANCELING A COMMAND

- If you decide you don't want to execute a command that you are in the process of entering, cancel it by typing \c

```
mysql> SELECT
    -> USER()
    -> \c
mysql>
```

CREATING A DATABASE

- A database is a container of data. It stores contacts, vendors, customers or any kind of data that you can think of.
- In MySQL, a database is a collection of objects that are used to store and manipulate data such as tables, database views, triggers, stored procedures, etc.

- To create a database in MySQL, you use the CREATE DATABASE statement as follows:

- `mysql>` CREATE DATABASE [IF NOT EXISTS] database_name;
- E.g
- *Mysql> create database [if not exists] test*

USING A DATABASE

- To get started on your own database, first check which databases currently exist.
- Use the SHOW statement to find out which databases currently exist on the server:

```
mysql> show databases;
+-----------+
| Database |
+-----------+
| mysql     |
| test      |
+-----------+
2 rows in set (0.01 sec)
```

USING A DATABASE

- To create a new database, issue the “create database” command:
  - `mysql> create database webdb;`
- To the select a database, issue the “use” command:
  - `mysql> use webdb;`

CREATING A TABLE

- Once you have selected a database, you can view all database tables:

```
mysql> show tables;
Empty set (0.02 sec)
```

- An empty set indicates that I have not created any tables yet.

CREATING A TABLE

- Let's create a table for storing pets.
- Table: pets
  - name: VARCHAR(20)
  - owner: VARCHAR(20)
  - species: VARCHAR(20)
  - sex: CHAR(1)
  - birth: DATE
  - date: DATE

VARCHAR is usually used to store string data.

CREATING A TABLE

- To create a table, use the CREATE TABLE command:

```
mysql> CREATE TABLE pet (
    -> name VARCHAR(20),
    -> owner VARCHAR(20),
    -> species VARCHAR(20),
    -> sex CHAR(1),
    -> birth DATE, death DATE);
Query OK, 0 rows affected (0.04 sec)
```

SHOWING TABLES

- To verify that the table has been created:

```
mysql> show tables;
+-------------------+
| Tables_in_test    |
+-------------------+
| pet               |
+-------------------+
1 row in set (0.01 sec)
```

DESCRIBING TABLES

- To view a table structure, use the DESCRIBE command:

```
mysql> describe pet;
+---------+-------------+------+-----+---------+-------+
| Field   | Type        | Null | Key | Default | Extra |
+---------+-------------+------+-----+---------+-------+
| name    | varchar(20) | YES  |     | NULL    |       |
| owner   | varchar(20) | YES  |     | NULL    |       |
| species | varchar(20) | YES  |     | NULL    |       |
| sex     | char(1)     | YES  |     | NULL    |       |
| birth   | date        | YES  |     | NULL    |       |
| death   | date        | YES  |     | NULL    |       |
+---------+-------------+------+-----+---------+-------+
6 rows in set (0.02 sec)
```

LOADING DATA

- Use the INSERT statement to enter data into a table.
- For example:

```
Mysql > INSERT INTO pet VALUES
  ('Fluffy','Harold','cat','f',
  '1999-02-04',NULL);
```

- The next slide shows a full set of sample data.

MORE DATA...

name	owner	species	sex	birth	death
Fluffy	Harold	cat	f	1993-02-04	
Claws	Gwen	cat	m	1994-03-17	
Buffy	Harold	dog	f	1989-05-13	
Fang	Benny	dog	m	1990-08-27	
Bowser	Diane	dog	m	1998-08-31	1995-07-29
Chirpy	Gwen	bird	f	1998-09-11	
Whistler	Gwen	bird		1997-12-09	
Slim	Benny	snake	m	1996-04-29	

LOADING SAMPLE DATA

- You could create a text file ‘pet.txt’ containing one record per line.
- Values must be separated by tabs, and given in the order in which the columns were listed in the CREATE TABLE statement.
- Then load the data via the LOAD DATA Command.

SAMPLE DATA FILE

Fluffy	Harold	cat	f	1993-02-04	\N
Claws	Gwen	cat	m	1994-03-17	\N
Buffy	Harold	dog	f	1989-05-13	\N
Fang	Benny	dog	m	1990-08-27	\N
Bowser	Diane	dog	m	1979-08-31	1995-07-29
Chirpy	Gwen	bird	f	1998-09-11	\N
Whistler	Gwen	bird	\N	1997-12-09	\N
Slim	Benny	snake	m	1996-04-29	\N

To Load pet.txt:

```
mysql>  LOAD DATA LOCAL INFILE "pet.txt" INTO TABLE pet;
```

FOR EACH OF THE EXAMPLES, ASSUME THE FOLLOWING SET OF DATA.

name	owner	species	sex	birth	death
Fluffy	Harold	cat	f	1993-02-04	
Claws	Gwen	cat	m	1994-03-17	
Buffy	Harold	dog	f	1989-05-13	
Fang	Benny	dog	m	1990-08-27	
Bowser	Diane	dog	m	1998-08-31	1995-07-29
Chirpy	Gwen	bird	f	1998-09-11	
Whistler	Gwen	bird		1997-12-09	
Slim	Benny	snake	m	1996-04-29	

SQL SELECT

- The SELECT statement is used to pull information from a table.
- The general format is:

```
SELECT what_to_select
FROM which_table
WHERE conditions_to_satisfy
```

SELECTING ALL DATA

- The simplest form of SELECT retrieves everything from a table

```
mysql> select * from pet;
+----------+--------+---------+------+------------+------------+
| name     | owner  | species | sex  | birth      | death      |
+----------+--------+---------+------+------------+------------+
| Fluffy   | Harold | cat     | f    | 1999-02-04 | NULL       |
| Claws    | Gwen   | cat     | f    | 1994-03-17 | NULL       |
| Buffy    | Harold | dog     | f    | 1989-05-13 | NULL       |
| Fang     | Benny  | dog     | m    | 1999-08-27 | NULL       |
| Bowser   | Diane  | dog     | m    | 1998-08-31 | 1995-07-29 |
| Chirpy   | Gwen   | bird    | f    | 1998-09-11 | NULL       |
| Whistler | Gwen   | bird    |      | 1997-12-09 | NULL       |
| Slim     | Benny  | snake   | m    | 1996-04-29 | NULL       |
+----------+--------+---------+------+------------+------------+
8 rows in set (0.00 sec)
```

SELECTING PARTICULAR ROWS

- You can select only particular rows from your table.
- For example, if you want to verify the change that you made to Bowser's birth date, select Bowser's record like this:

```
mysql> SELECT * FROM pet WHERE name = "Bowser";

+--------+-------+---------+------+------------+------------+
| name   | owner | species | sex  | birth      | death      |
+--------+-------+---------+------+------------+------------+
| Bowser | Diane | dog     |  m   | 1998-08-31 | 1995-07-29 |
+--------+-------+---------+------+------------+------------+
1 row in set (0.00 sec)
```

SELECTING PARTICULAR ROWS

- To find all animals born after 1998

```
SELECT * FROM pet WHERE birth >= "1998-1-1";
```

- To find all female dogs, use a logical AND

```
SELECT * FROM pet WHERE species = "dog" AND sex = "f";
```

- To find all snakes or birds, use a logical OR

```
SELECT * FROM pet WHERE species = "snake"
OR species = "bird";
```

SELECTING PARTICULAR COLUMNS

- If you don't want to see entire rows from your table, just name the columns in which you are interested, separated by commas.

- For example, if you want to know when your pets were born, select the name and birth columns. *(see example next slide)*

SELECTING PARTICULAR COLUMNS

```
mysql> select name, birth from pet;
+-----------+-------------+
| name      | birth       |
+-----------+-------------+
| Fluffy    | 1999-02-04 |
| Claws     | 1994-03-17 |
| Buffy     | 1989-05-13 |
| Fang      | 1999-08-27 |
| Bowser    | 1998-08-31 |
| Chirpy    | 1998-09-11 |
| Whistler | 1997-12-09 |
| Slim      | 1996-04-29 |
+-----------+-------------+
8 rows in set (0.01 sec)
```

SORTING DATA

- To sort a result, use an ORDER BY clause.
- For example, to view animal birthdays, sorted by date:

```
mysql> SELECT name, birth FROM pet ORDER BY birth;

+----------+------------+
| name     | birth      |
+----------+------------+
| Buffy    | 1989-05-13 |
| Claws    | 1994-03-17 |
| Slim     | 1996-04-29 |
| Whistler | 1997-12-09 |
| Bowser   | 1998-08-31 |
| Chirpy   | 1998-09-11 |
| Fluffy   | 1999-02-04 |
| Fang     | 1999-08-27 |
+----------+------------+
8 rows in set (0.02 sec)
```

SORTING DATA

- To sort in reverse order, add the DESC (descending keyword)

```
mysql> SELECT name, birth FROM pet ORDER BY birth DESC;

+----------+------------+
| name     | birth      |
+----------+------------+
| Fang     | 1999-08-27 |
| Fluffy   | 1999-02-04 |
| Chirpy   | 1998-09-11 |
| Bowser   | 1998-08-31 |
| Whistler | 1997-12-09 |
| Slim     | 1996-04-29 |
| Claws    | 1994-03-17 |
| Buffy    | 1989-05-13 |
+----------+------------+
8 rows in set (0.02 sec)
```

WORKING WITH NULLS

- NULL means missing value or unknown value.
- To test for NULL, you cannot use the arithmetic comparison operators, such as =, < or <>.
- Rather, you must use the IS NULL and IS NOT NULL operators instead.

WORKING WITH NULLS

- For example, to find all your dead pets (what a morbid example!)

```
mysql> select name from pet where death
     > IS NOT NULL;
+---------+
| name    |
+---------+
| Bowser |
+---------+
1 row in set (0.01 sec)
```

PATTERN MATCHING

- MySQL provides:
  - standard SQL pattern matching; and
  - regular expression pattern matching, similar to those used by Unix utilities such as vi, grep and sed.
- SQL Pattern matching:
  - To perform pattern matching, use the **LIKE** or **NOT LIKE** comparison operators
  - By default, patterns are case insensitive.
- Special Characters:
  - **_** Used to match any single character.
  - **%** Used to match an arbitrary number of characters.

PATTERN MATCHING EXAMPLE

- To find names beginning with ‘b’:

```
mysql> SELECT * FROM pet WHERE name LIKE "b%";
+--------+--------+---------+------+------------+------------+
| name   | owner  | species | sex  | birth      | death      |
+--------+--------+---------+------+------------+------------+
| Buffy  | Harold | dog     | f    | 1989-05-13 | NULL       |
| Bowser | Diane  | dog     | m    | 1989-08-31 | 1995-07-29 |
+--------+--------+---------+------+------------+------------+
```

PATTERN MATCHING EXAMPLE

- To find names ending with `fy':

```
mysql> SELECT * FROM pet WHERE name LIKE "%fy";
+--------+--------+---------+------+------------+-------+
| name   | owner  | species | sex  | birth      | death |
+--------+--------+---------+------+------------+-------+
| Flufffy | Harold | cat     | f    | 1993-02-04 | NULL  |
| Buffy  | Harold | dog     | f    | 1989-05-13 | NULL  |
+--------+--------+---------+------+------------+-------+
```

PATTERN MATCHING EXAMPLE

- To find names containing a ‘w’:

```
mysql> SELECT * FROM pet WHERE name LIKE "%w%";
+----------+-------+---------+------+------------+------------+
| name     | owner | species | sex  | birth      | death      |
+----------+-------+---------+------+------------+------------+
| Claws    | Gwen  | cat     | m    | 1994-03-17 | NULL       |
| Bowser   | Diane | dog     | m    | 1989-08-31 | 1995-07-29 |
| Whistler | Gwen  | bird    | NULL | 1997-12-09 | NULL       |
+----------+-------+---------+------+------------+------------+
```

PATTERN MATCHING EXAMPLE

- To find names containing exactly five characters, use the _ pattern character:

```
mysql> SELECT * FROM pet WHERE name LIKE "_ _ _ _ _";
+-------+--------+---------+------+------------+-------+
| name  | owner  | species | sex  | birth      | death |
+-------+--------+---------+------+------------+-------+
| Claws | Gwen   | cat     | m    | 1994-03-17 | NULL  |
| Buffy | Harold | dog     | f    | 1989-05-13 | NULL  |
+-------+--------+---------+------+------------+-------+

Note :there should not be a any space in between _ _ _ _ _
```

REGULAR EXPRESSIONS

- Some characteristics of extended regular expressions are:
  - . matches any single character.
  - A character class [...] matches any character within the brackets. For example, [abc] matches a, b, or c. To name a range of characters, use a dash. [a-z] matches any lowercase letter, whereas [0-9] matches any digit.
  - * matches zero or more instances of the thing preceding it. For example, x* matches any number of x characters, [0-9]* matches any number of digits, and .* matches any number of anything.
  - To anchor a pattern so that it must match the beginning or end of the value being tested, use ^ at the beginning or $ at the end of the pattern.

REG EX EXAMPLE

- To find names beginning with b, use ^ to match the beginning of the name:

```
mysql> SELECT * FROM pet WHERE name REGEXP "^b";
+---------+--------+---------+------+------------+------------+
| name    | owner  | species | sex  | birth      | death      |
+---------+--------+---------+------+------------+------------+
| Buffy   | Harold | dog     | f    | 1989-05-13 | NULL       |
| Bowser  | Diane  | dog     | m    | 1989-08-31 | 1995-07-29 |
+---------+--------+---------+------+------------+------------+
```

REG EX EXAMPLE

- To find names ending with `fy', use `$' to match the end of the name:

```
mysql> SELECT * FROM pet WHERE name REGEXP "fy$";
+--------+--------+---------+------+------------+-------+
| name   | owner  | species | sex  | birth      | death |
+--------+--------+---------+------+------------+-------+
| Fluffy | Harold | cat     | f    | 1993-02-04 | NULL  |
| Buffy  | Harold | dog     | f    | 1989-05-13 | NULL  |
+--------+--------+---------+------+------------+-------+
```

COUNTING ROWS

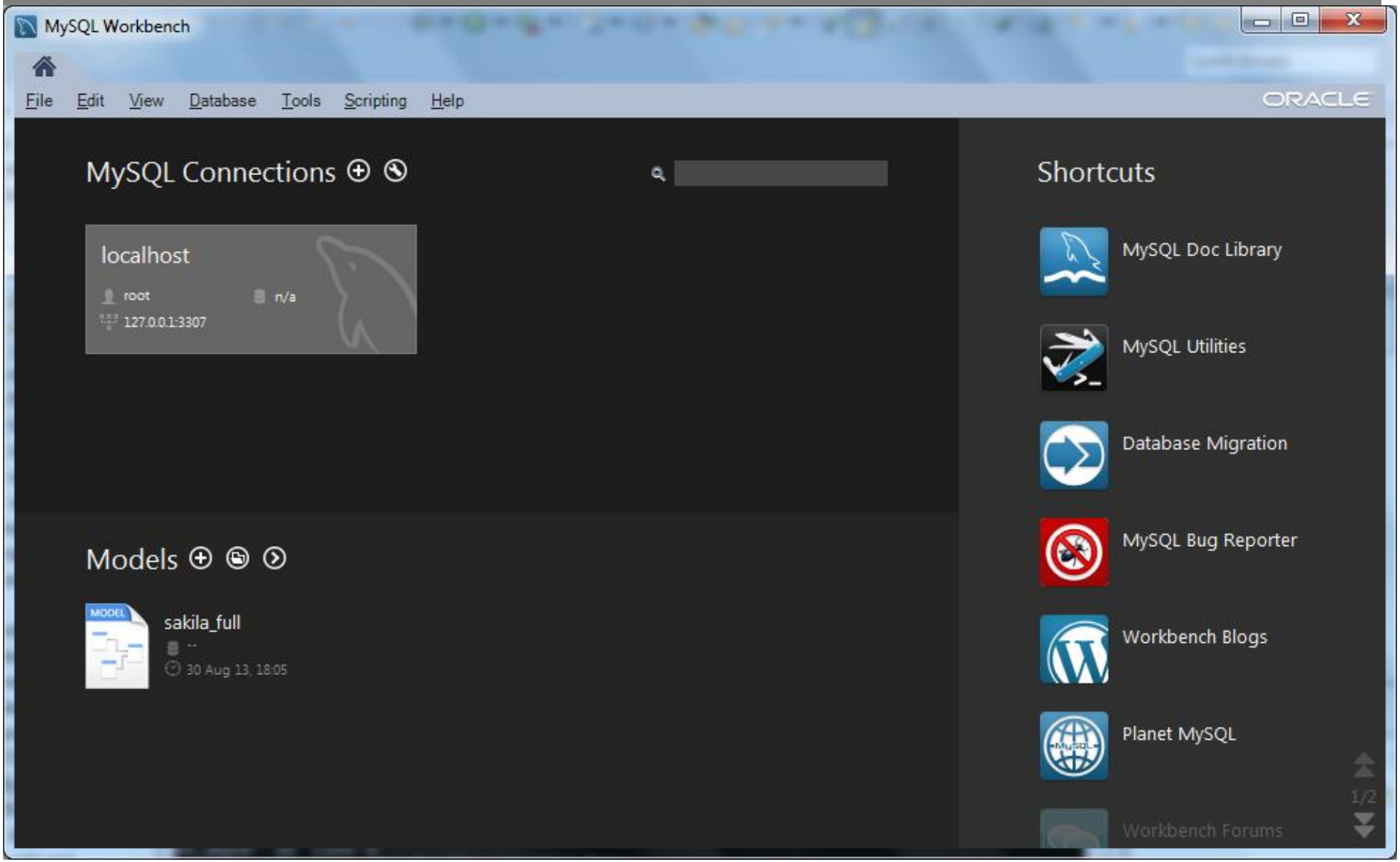
- Databases are often used to answer the question, "How often does a certain type of data occur in a table?"
- For example, you might want to know how many pets you have, or how many pets each owner has.
- Counting the total number of animals you have is the same question as “How many rows are in the pet table?” because there is one record per pet.
- The COUNT() function counts the number of non-NULL results.

COUNTING ROWS EXAMPLE

- A query to determine total number of pets:

```
mysql> SELECT COUNT(*) FROM pet;
+----------+
| COUNT(*) |
+----------+
|        9 |
+----------+
```

WORKBENCH


MySQL Workbench
File
Edit
View
Database
Tools
Scripting
Help
ORACLE
MySQL Connections
localhost
root
n/a
127.0.0.1:3307
Models
MODEL
sakila_full
--
30 Aug 13, 18:05
Shortcuts
MySQL Doc Library
MySQL Utilities
Database Migration
MySQL Bug Reporter
Workbench Blogs
Planet MySQL
1/2
Workbench Forums

MySQL Workbench

localhost

File Edit View Query Database Server Tools Scripting Help

ORACLE

Navigator

MANAGEMENT

- Server Status
- Client Connections
- Users and Privileges
- Status and System Variables
- Data Export
- Data Import/Restore

INSTANCE

- Startup / Shutdown
- Server Logs
- Options File

SCHEMAS

Filter objects

- test
  - Tables
    - accounts
    - books
  - Views
  - Stored Procedures

Information

No object selected

Object Info | Session

Query 1

```
1  accountsaccounts
```

SQL Additions

SELECT

Topic: SELECT

Syntax:

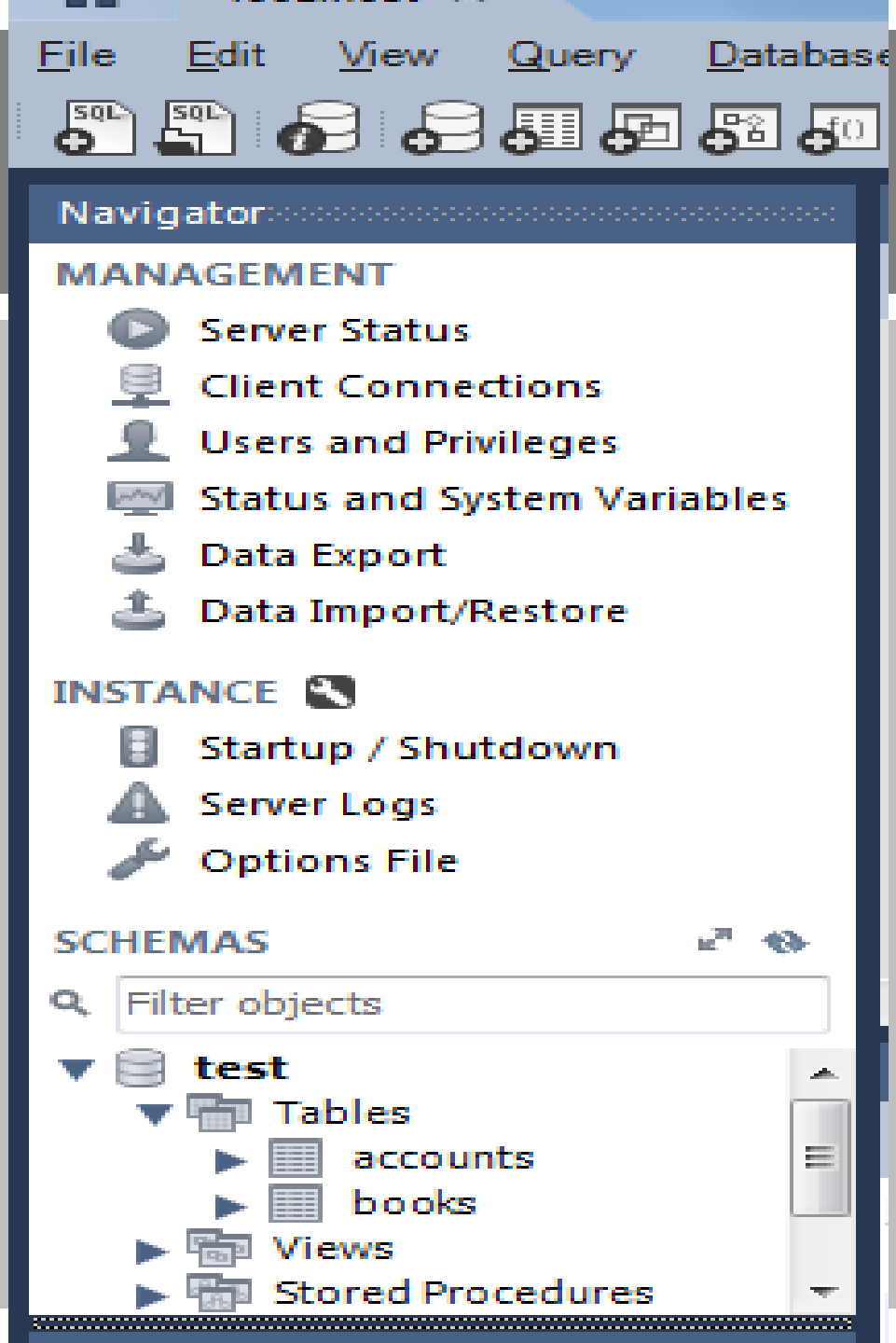
```
SELECT
    [ALL | DISTINCT | DISTINCTROW ]
      [HIGH_PRIORITY]
      [STRAIGHT_JOIN]
      [SQL_SMALL_RESULT] [SQL_BIG_RESULT] [
RESULT]
      [SQL_CACHE | SQL_NO_CACHE] [SQL_CALC_
    select_expr [, select_expr ...]
    [FROM table_references
      [PARTITION partition_list]
    [WHERE where_condition]
    [GROUP BY {col_name | expr | position}
      [ASC | DESC], ... [WITH ROLLUP]]
    [HAVING where_condition]
```

Context Help | Snippets

Output

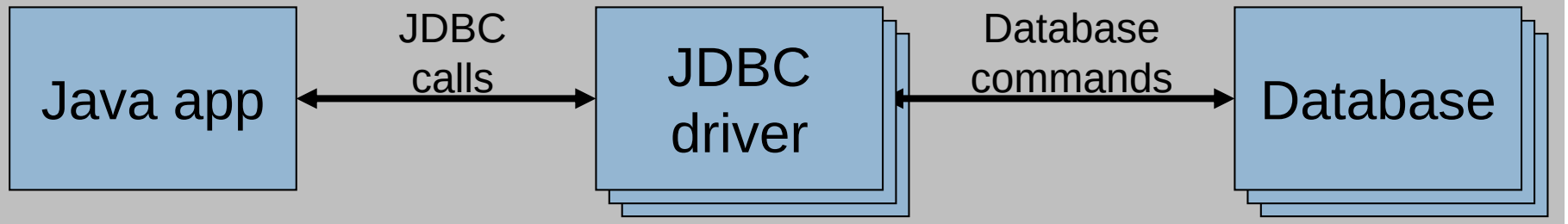
Action Output

	Time	Action	Message	Duration / Fetch
21	00:17:35	ANALYZE TABLE `test`.`accounts`	OK	0.000 sec
22	00:17:38	SET GLOBAL innodb_optimize_fulltext_only=0, innod...	OK	0.000 sec
23	00:17:38	OPTIMIZE TABLE `test`.`accounts`	OK	0.000 sec
24	00:17:39	SET GLOBAL innodb_optimize_fulltext_only=True	OK	0.000 sec
25	04:29:53	SELECT * FROM test.accounts LIMIT 0, 1000	1 row(s) returned	0.000 sec / 0.000 sec
26	04:30:06	Commit recordset changes	Commit complete	


File
Edit
View
Query
Database
Navigator
MANAGEMENT
Server Status
Client Connections
Users and Privileges
Status and System Variables
Data Export
Data Import/Restore
INSTANCE
Startup / Shutdown
Server Logs
Options File
SCHEMAS
Filter objects
test
Tables
accounts
books
Views
Stored Procedures

JAVA DATABASE CONNECTIVITY (JDBC)

- An interface to communicate with a relational database
  - Allows database agnostic Java code
  - Treat database tables/rows/columns as Java objects
- JDBC driver
  - An implementation of the JDBC interface
  - Communicates with a particular database



ECLIPSE JDBC SETUP

- Install driver
  - Download MySQL JDBC driver from assignment Web page
  - Unzip mysql-connector-xxx.jar
  - Add mysql-connector-xxx.jar to Eclipse project
    - *Project → Properties → Java Build Path → Libraries → Add External JARs*
  - Download from *http://dev.mysql.com/downloads/connector/*

JDBC STEPS

- Connect to database
- Query data (or Insert/update/delete)
- Process results
- Close connection to database

1. CONNECT TO DATABASE

- **Load JDBC driver**
  - `Class.forName("com.mysql.jdbc.Driver").newInstance();`
- **Make connection**
  - `Connection conn = DriverManager.getConnection(url);`
- **URL**
  - **Format: "jdbc:<*subprotocol*>:<*subname*>"**
  - `jdbc:mysql://128.100.53.33/GROUPNUMBER?user=USER&password=PASSWORD`

2. QUERY DATABASE

a. **Create statement**

- `Statement stmt = conn.createStatement();`
- stmt object sends SQL commands to database
- Methods
  - executeQuery() for SELECT statements
  - executeUpdate() for INSERT, UPDATE, DELETE, statements

b. **Send SQL statements**

- stmt.executeQuery("SELECT …");
- stmt.executeUpdate("INSERT …");

3. PROCESS RESULTS

- Result of a SELECT statement (rows/columns) returned as a ResultSet object
  - `ResultSet rs = stmt.executeQuery("SELECT * FROM users");`
- Step through each row in the result
  - `rs.next()`
- Get column values in a row
  - `String userid = rs.getString("userid");`
  - `int type = rs.getInt("type");`

users table				
userid	**firstname**	**lastname**	**password**	**type**
Bob	Bob	King	cat	0
John	John	Smith	pass	1

PRINT THE USERS TABLE

```
ResultSet rs = stmt.executeQuery("SELECT * FROM
      users");

while (rs.next()) {
   String userid = rs.getString(1);
   String firstname = rs.getString("firstname");
   String lastname = rs.getString("lastname");
   String password = rs.getString(4);
   int type = rs.getInt("type");
   System.out.println(userid + " " + firstname + "
      " + lastname + " " + password + " " + type);
}
```

users table				
userid	**firstname**	**lastname**	**password**	**type**
Bob	Bob	King	cat	0
John	John	Smith	pass	1

ADD A ROW TO THE USERS TABLE

```
String str =
     "INSERT INTO users
      VALUES('Bob', 'Bob', 'King',
             'cat', 0)";

// Returns number of rows in table
int rows = stmt.executeUpdate(str);
```

users table				
userid	**firstname**	**lastname**	**password**	**type**
Bob	Bob	King	cat	0

4. CLOSE CONNECTION TO DATABASE

- Close the ResultSet object
  - `rs.close();`
- Close the Statement object
  - `stmt.close();`
- Close the connection
  - `conn.close();`

```
import java.sql.*;

public class Tester {
  public static void main(String[] args) {
     try {
        // Load JDBC driver
        Class.forName("com.mysql.jdbc.Driver").newInstance();

        // Make connection
        String url =

     "jdbc:mysql://128.100.53.33/GRP?user=USER&password=PASS"
        Connection conn = DriverManager.getConnection(url);

        // Create statement
        Statement stmt = conn.createStatement();

        // Print the users table
        ResultSet rs = stmt.executeQuery("SELECT * FROM
  users");
        while (rs.next()) {
              ...
        }

        // Cleanup
        rs.close(); stmt.close(); conn.close();

  } catch (Exception e) {
        System.out.println("exception " + e);
  }
}
```

TRANSACTIONS

- Currently every executeUpdate() is “finalized” right away
- Sometimes want to a set of updates to all fail or all succeed
  - E.g. add to Appointments and Bookings tables
  - Treat both inserts as one transaction
- Transaction
  - Used to group several SQL statements together
  - Either all succeed or all fail

TRANSACTIONS

- Commit
  - Execute all statements as one unit
  - “Finalize” updates
- Rollback
  - Abort transaction
  - All uncommited statements are discarded
  - Revert database to original state

TRANSACTIONS IN JDBC

- Disable auto-commit for the connection
  - conn.setAutoCommit(false);
- Call necessary executeUpdate() statements
- Commit or rollback
  - conn.commit();
  - conn.rollback();

REFERENCES

- JDBC API Documentation
  - http://docs.oracle.com/javase/7/docs/technotes/guides/jdbc/
  - Note: this is a newer JDBC API, but should be mostly compatible