Introduction to MySQL

Acknowledge

These slides are a modified version of the slides available online at

- https://www.google.it/url?sa=t&rct=j&q=&esrc=s&source=web&cd=1&cad=rja&uact=8&ved=0ahUKEwiYlv6D4rfJAhWFORoKHaqlBVgQFgghMAA&url=http%3A%2F%2Fmiageprojet2.unice.fr%2F%40api%2Fdeki%2Ffiles%2F1830%2F%3DIntro_to_MySQL.ppt&usg=AFQjCNGzdn0C4HYp3JkSNY1_A1T2MSRGfg&sig2=MNbk3ujw91NxZ4vLN_750w
- http://www.slideshare.net/chauhantushar/introduction-to-mysql/4

Road Map

- 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 of the information comes from Chapter 3, MySQL Tutorial:
  - http://dev.mysql.com/doc/refman/5.7/en/tutorial.html
    - http://www.mysql.com/documentation/mysql/bychapter/manual_Tutorial.html#Tutorial

Introduction to Mysql

MySQL

- Officially pronounced "my Ess Que Ell" (not my sequel)
- MySQL is a very popular, open source DBMS
- MySQL databases are relational
- Handles very large databases;
- very fast performance; reliable.
- MySQL is compatible with standard SQL
- Why are we using MySQL?
  - Free (much cheaper than Oracle!)
  - Each student can install MySQL locally.
  - Multi-user access to a number of databases offered
  - Easy to use Shell for creating tables, querying tables, etc.
  - Easy to use with Java JDBC
  - MySQL is frequently used by PHP and Perl
  - Commercial version of MySQL is also provided (including technical support)

History of MySQL

- Founded and developed by David Axmark, Allan Larsson, and Michael "Monty" Widenius
- Named after Monty's daughter, *My*
- MySQL Dolphin logo is "Sakila", the name of a town in Arusha, Tanzania
- Written in C and C++
- Works on many different platforms
- Sun acquired MySQL AB in Jan 2008 for $1 billion dollars

MySQL Products Overview

MySQL Server

- Community Server
- Enterprise Server
- Embedded Server
- Cluster (Standard and Carrier-Grade)

MySQL GUI Tools

- Query Browser
- Administrator
- Migration Toolkit
- Visual Studio Plug-in
- MySQL Workbench (New!)

MySQL Drivers

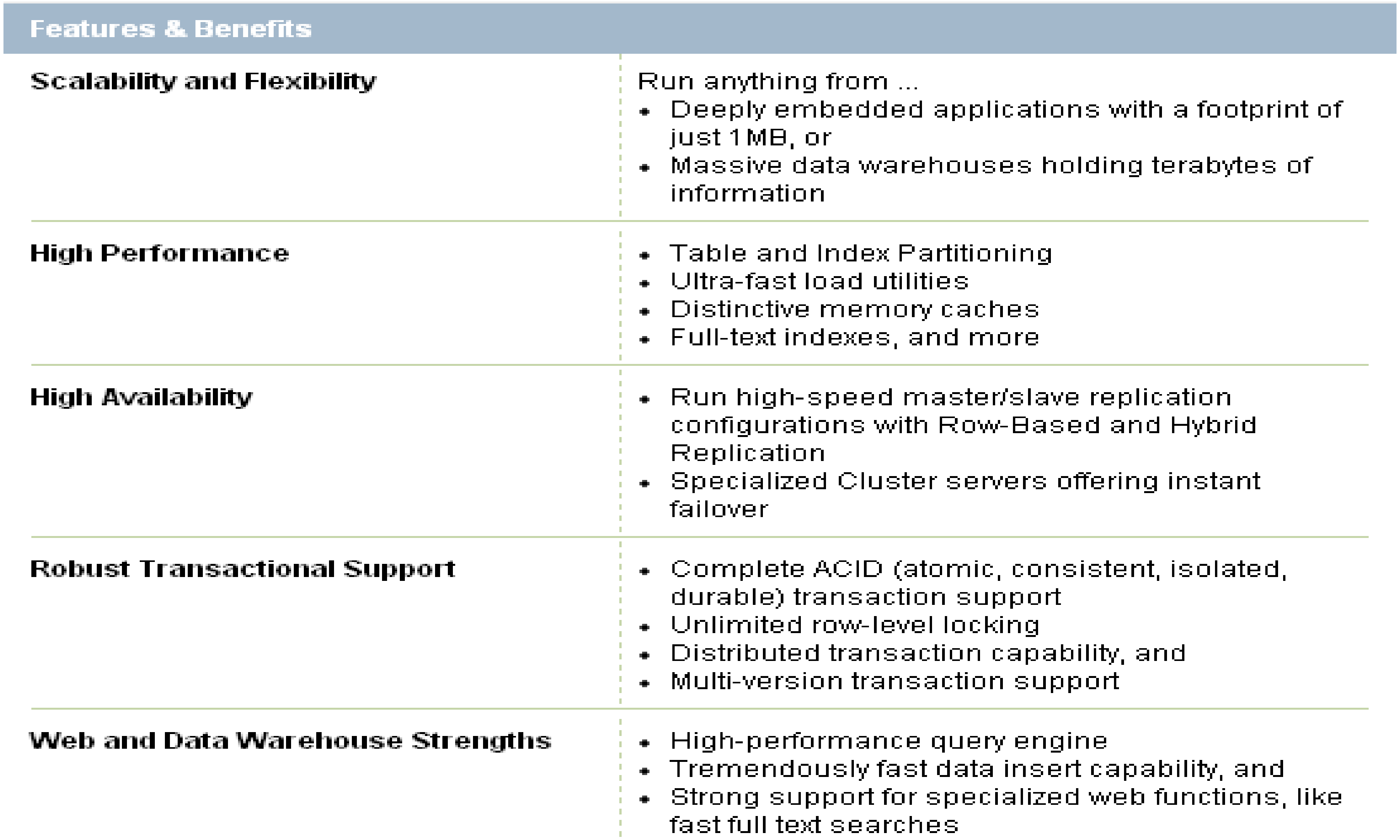
- JDBC
- ODBC
- .NET
- PHP

MySQL: characteristics

MySQL Server works in:

- client/server systems → a system consisting of a multi-threaded SQL server that supports different backends, several different client programs and libraries, administrative tools, and a wide range of application programming interfaces (APIs)
- embedded systems → provide MySQL Server as an embedded multi-threaded library that can be linked into an application to get a smaller, faster, easier-to-manage standalone product.

MySQL: Features & Benefits...

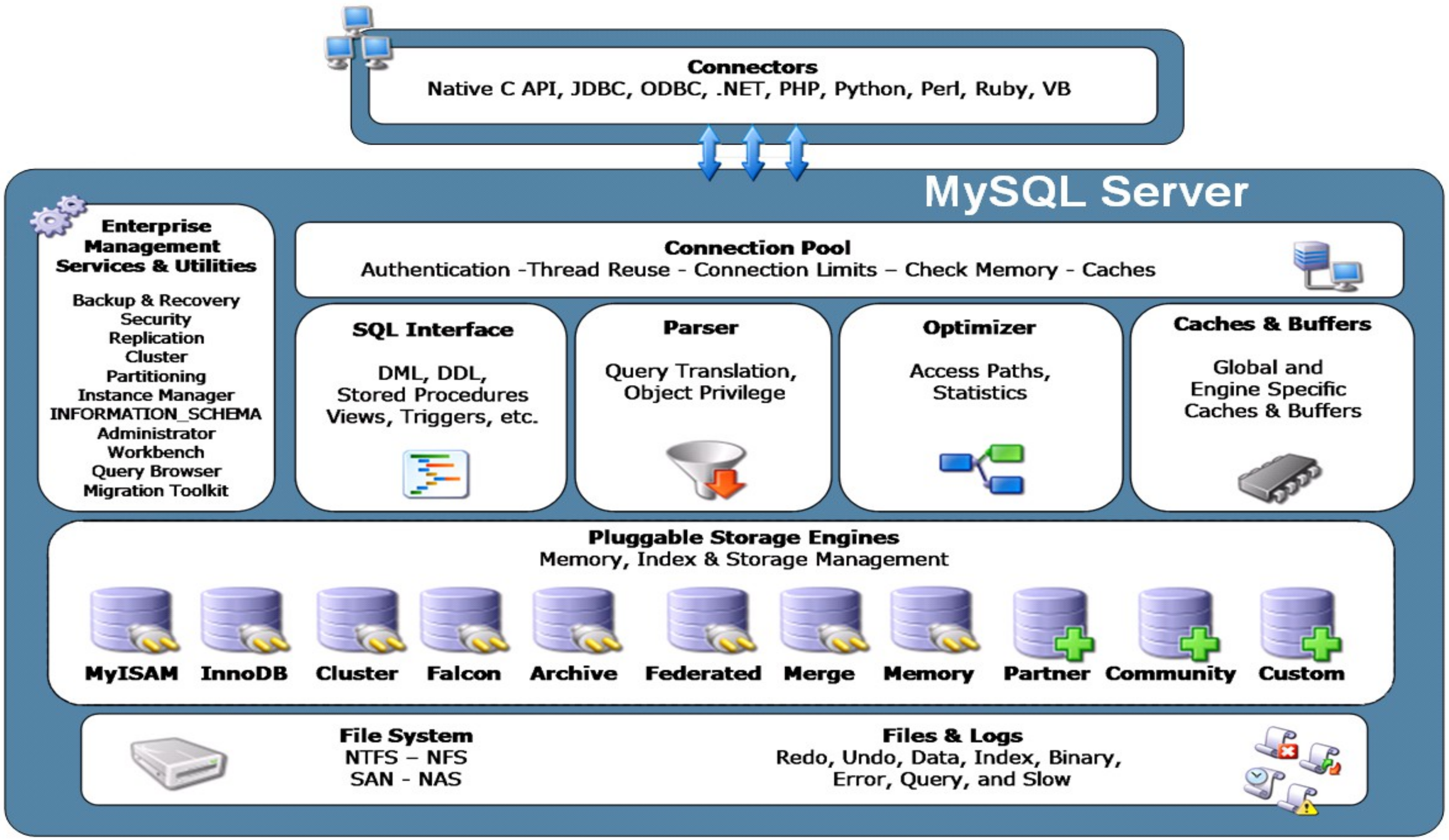


Features & Benefits	
Scalability and Flexibility	Run anything from ... • Deeply embedded applications with a footprint of just 1MB, or • Massive data warehouses holding terabytes of information
High Performance	• Table and Index Partitioning • Ultra-fast load utilities • Distinctive memory caches • Full-text indexes, and more
High Availability	• Run high-speed master/slave replication configurations with Row-Based and Hybrid Replication • Specialized Cluster servers offering instant failover
Robust Transactional Support	• Complete ACID (atomic, consistent, isolated, durable) transaction support • Unlimited row-level locking • Distributed transaction capability, and • Multi-version transaction support
Web and Data Warehouse Strengths	• High-performance query engine • Tremendously fast data insert capability, and • Strong support for specialized web functions, like fast full text searches

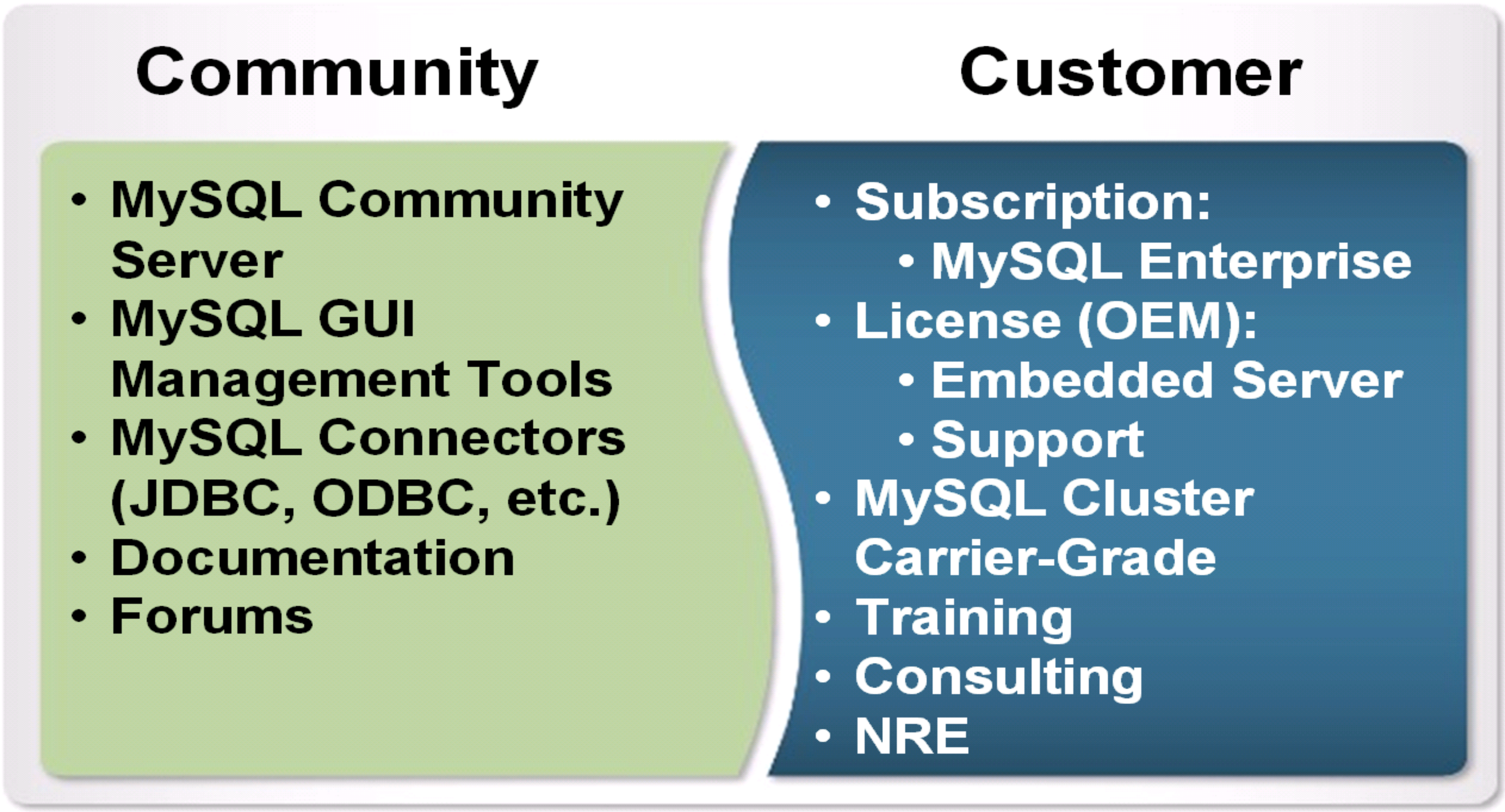
...MySQL: Features & Benefits

Strong Data Protection	• Powerful mechanisms for ensuring only authorized users have access • SSH and SSL support safe and secure connections • Powerful data encryption and decryption functions
Comprehensive Application Development	• Support for stored procedures, triggers, functions, views, cursors, ANSI-standard SQL, and more • Plug-in libraries to embed MySQL database support into nearly any application
Management Ease	• Use Event Scheduler automatically schedule common recurring SQL-based tasks to execute on the database server. • Average time from software download to complete installation is less than fifteen minutes.
Open Source Freedom and 24 x 7 Support	• Around-the-clock support and indemnification available through MySQL Network • Enterprise quality and enterprise ready, from installation to support
Lowest Total Cost of Ownership	• Save on database licensing costs and hardware expenditures, all while cutting systems downtime

MySQL: Architecture


Connectors
Native C API, JDBC, ODBC, .NET, PHP, Python, Perl, Ruby, VB
MySQL Server
Enterprise Management Services & Utilities
Backup & Recovery
Security
Replication
Cluster
Partitioning
Instance Manager
INFORMATION_SCHEMA
Administrator
Workbench
Query Browser
Migration Toolkit
Connection Pool
Authentication -Thread Reuse - Connection Limits – Check Memory - Caches
SQL Interface
DML, DDL, Stored Procedures Views, Triggers, etc.
Parser
Query Translation, Object Privilege
Optimizer
Access Paths, Statistics
Caches & Buffers
Global and Engine Specific Caches & Buffers
Pluggable Storage Engines
Memory, Index & Storage Management
MyISAM
InnoDB
Cluster
Falcon
Archive
Federated
Merge
Memory
Partner
Community
Custom
File System
NTFS – NFS
SAN - NAS
Files & Logs
Redo, Undo, Data, Index, Binary, Error, Query, and Slow

MySQL: Community & Customers



Community

- MySQL Community Server
- MySQL GUI Management Tools
- MySQL Connectors (JDBC, ODBC, etc.)
- Documentation
- Forums

Customer

- Subscription:
  - MySQL Enterprise
- License (OEM):
  - Embedded Server
  - Support
- MySQL Cluster Carrier-Grade
- Training
- Consulting
- NRE

MySQL: Workbench

MySQL Workbench enables:

- a DBA, developer, or data architect
- to visually design, generate, and manage all types of databases
    - including Web, OLTP, and data warehouse databases
    - It includes everything a data modeler needs for creating complex ER models, and also delivers key features for performing difficult change management and documentation tasks that are normally time consuming

MySQL: Workbench

Some characteristics

- ❖ **Forward and Reverse Engineering**
  - A visual data model can easily be transformed into a physical database on a target MySQL Server with just a few mouse clicks.
  - it can also import SQL scripts to build models and export models to DDL scripts that can be run at a later time.
- ❖ **Change Management**

MySQL Workbench

- **Database Documentation**
  - Documenting database designs can be a time-consuming process.
    - MySQL Workbench includes DBDoc that enables a DBA or developer to deliver point-and-click database documentation.
  - Models can be documented in either HTML or plain text format, and includes all the objects and models in a current MySQL Workbench session

Resources

- General starting point
  - > http://www.mysql.com/
- Developer focused
  - > http://dev.mysql.com/

Installing MySQL

MySQL: Installation

Instruction available at Ch. 2 of the MySQL tutorial available at http://dev.mysql.com/doc/refman/5.7/en/tutorial.html

Connecting and Disconnetting to/from MySQL

Conventions

- commands meant to be executed within a particular shell,for example, shell>
  - root-shell> is similar but should be executed as root
- mysql> indicates a statement that has to be executed from the mysql client program
- SQL keywords are not case sensitive

Connecting to MySQL

- MySQL provides an interactive shell for creating tables, inserting data, etc.
- On Windows, just go to c:\mysql\bin, and type:
  - **`Mysql`** or
  - mysql -u *user* -p;
- Or, click on the Windows icon

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
```

Connecting to MySQL

How to use the mysql client

- **mysql** is an interactive program that enables you to:
  - connect to a MySQL server,
  - Run queries,
  - view the results
- mysql may also be used in batch mode:
  - *place your queries in a file beforehand*, then tell mysql to execute the contents of the file
- To see a list of options provided by mysql
  - shell> mysql --help

Entering & Editing commands

- Prompt mysql>
  - issue a command
  - Mysql sends it to the server for execution
  - displays the results
  - prints another mysql>
- a command could span multiple lines
- A command normally consists of SQL statement followed by a semicolon

MySQL commands

- help \h
- Quit/exit \q
- Cancel the command \c
- Change database use
- …etc

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:

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

Basic Queries

- Here's another query. It demonstrates that you can use mysql as a simple calculator:

```
mysql> SELECT SIN(PI()/4), (4+1)*5;
+-------------+---------+
| SIN(PI()/4) | (4+1)*5 |
+-------------+---------+
|    0.707107 |      25 |
+-------------+---------+
```

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   |
+--------------------+--------------+
```

Command prompt

prompt	meaning
mysql>	Ready for new command.
->	Waiting for next line of multiple-line command.
‘>	Waiting for next line, waiting for completion of a string that began with a single quote (“'”).
“>	Waiting for next line, waiting for completion of a string that began with a double quote (“"”).
` >	Waiting for next line, waiting for completion of an identifier that began with a backtick (“ ` ”).
/*>	Waiting for next line, waiting for completion of a comment that began with /*.

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, Removing and Getting Information for a Database

Info about DBs and tables

- Listing the databases on the MySQL server host
  - `mysql>`show databases;
- Access/change database
  - `mysql>`Use [database_name];
- Showing the current selected database
  - `mysql>` select database();
- Showing tables in the current database
  - `mysql>`show tables;
- Showing the structure of a table
  - `mysql>` describe [table_name];

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 (and for which the user has privileges):

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

Using a Database

View Users (Before MySQL 5.7.6)

- mysql> SELECT User, Host, Password FROM mysql.user;

Changing password for a user

- After 5.7.6, use SET PASSWORD
  - mysql> ALTER USER user IDENTIFIED BY 'new_password';
  - Ex.: ALTER USER 'root'@'localhost' IDENTIFIED BY 'new_password';
- Before 5.7.6, use SET PASSWORD:
  - mysql> SET PASSWORD FOR user = PASSWORD('new_password');

Using a Databases

Create a user

- mysql> CREATE USER *user* [IDENTIFIED BY 'new-password'];
- See “help CREATE USER”

Remove a user

- mysql> DROP USER *user*;

Using a Database

- To create a new database, issue the "create database" command:
  - `mysql> create database webdb;`
  - Note: **<u>Database names are case sensitive</u>**
- To the select a database, issue the "use" command:
  - `mysql> use webdb;`
- To see what database is selected
  - `mysql> select database();`

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: pet
  - name: VARCHAR(20)
  - owner: VARCHAR(20)
  - species: VARCHAR(20)
  - sex: CHAR(1)
  - birth: DATE
  - death: 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)
```

Deleting a Table

- To delete an entire table, use the DROP TABLE command:

```
mysql> drop table pet;
Query OK, 0 rows affected (0.02 sec)
```

Loading Data

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

```
INSERT INTO pet VALUES
  ('Puffball','Diane','hamster','f',
  '1999-03-30',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 INFILE 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	

Manipulating Instances of Tables of a Database

Manipulating Table Instances

- Remove records of a table
  - `mysql> DELETE FROM tableName;`
  - `[WHERE where_condition];`
- Update records of a table
  - `UPDATE pet SET birth = '1989-08-31' WHERE name ='Bowser';`

Querying Tables of a Database

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)
```

Try the same select:

- without the last (“)
- without the last (“) and (;)
- Try with \c

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";
  ```

AND has higher precedence than OR → Use paranthesis if necessary

Selecting Particular Columns

- For having a selection of columns of a table, just name the columns you are interested in, separated by commas.
- **Example:** 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.
- Example: 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)
```

Sorting Data

- Sorting on multiple columns in different directions
  - Get name, species, birth with animals in ascending order and date (within animal type) in descending order (youngest first)
  - `mysql> SELECT name, species, birth FROM pet ORDER BY species, birth DESC;`
  - **Try the opposite**

Selecting Particular Rows

- Find out who owns pets

```
SELECT owner FROM pet;
```

- Find out who owns pets (without duplicate)

```
SELECT DISTINCT owner FROM pet;
```

- Get birth dates for male dogs and female cats

```
SELECT name, species, birth FROM pet WHERE (species = "dog" AND
    sex=”m”)  OR (species = "cat" AND sex=”f”);
```

Try with using union

```
SELECT name, species, birth FROM pet WHERE (species = "dog" AND
    sex=”m”)
UNION  SELECT name, species, birth FROM pet WHERE (species =
    "cat" AND sex=”f”);
```

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.

Working with NULLs

- Find all your dead pets

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

Working with NULLs

- Two NULL values are regarded as equal in a GROUP BY
  - Ex.: create a query with a group by on an attribute haing NULL values
- NULL values are presented:
  - first with ORDER BY … ASC
  - last with ORDER BY ... DESC

Pattern Matching

- MySQL provides:
  - standard SQL pattern matching
  - regular expression pattern matching
- 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

- Find names ending with `fy':

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

Pattern Matching Example

- 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

- 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  |
+-------+--------+---------+------+------------+-------+
```

Regular Expression Matching

- The other type of pattern matching provided by MySQL uses extended regular expressions.
- Testing for a match for this type of pattern, use the REGEXP and NOT REGEXP operators (or RLIKE and NOT RLIKE, which are synonyms).

Regular Expressions

- Some characteristics of extended regular expressions:
  - "." matches any **single** character.
  - A character class [...] matches any character within the brackets.
    - Example: [abc] matches a, b, or c.
    - To name a range of characters, use a dash.
      - [a-z] matches any lowercase letter
      - [0-9] matches any digit.
  - "*" matches zero or more instances of the thing preceding it.
    - Example: x* matches any number of x characters
    - [0-9]* matches any number of digits
    - .* 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 Expressions: Example

- 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 Expressions: Example

- 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 often used to answer the question,
  - "How often does a certain type of data occur in a table?"
  - Example:
    - 1) how many pets are stored
    - 2) 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 results

Counting Rows Example

- A query to determine total number of pets:

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

Counting Rows Example

- Finding how many pets each owner has:

```
mysql> SELECT owner, COUNT(*) FROM
  pet GROUP BY owner;
+----------+----------+
| OWNER    | COUNT(*) |
+----------+----------+
| Benny    |    2     |
+----------+----------+
| Diane    |    2     |
+----------+----------+
| Gwen     |    3     |
+----------+----------+
| Harold   |    2     |
+----------+----------+
```

Selecting Particular Rows

- Find out number of animals per species

```
SELECT species, count(*) FROM pet GROUP BY species;
```

- Find out number of animals per sex

```
SELECT sex, count(*) FROM pet GROUP BY sex;
```

- Find out number of animals per combination of species and sex

```
SELECT species, sex, count(*) FROM pet GROUP BY
species, sex;
```

Selecting Particular Rows

- Find out number of dogs and cats per combination of species and sex

  SELECT species, sex, count(*) FROM pet WHERE species = 'dog' or species = 'cat' GROUP BY species, sex;

  - Try what happens changing OR with AND

- Find out number of animals per combination of species and sex, only for animals whose sex is known

  SELECT species, sex, count(*) FROM pet WHERE sex IS NOT NULL GROUP BY species, sex;

Batch Mode

- MySQL used interactively
  - to enter queries and view the results.
- MySQL can be run in batch mode.
  - put the commands you want to run in a file
  - tell mysql to read its input from the file:
    - The created file is script file that is requested to be executed
- `shell> mysql < batch-file`
- `shell> mysql -t < batch-file`

Exercise 1

- Create a new DB or a table *shop* in an existing DB

```
CREATE TABLE shop (
article INT(4) UNSIGNED ZEROFILL DEFAULT '0000'
   NOT NULL,
dealer VARCHAR(20) DEFAULT '' NOT NULL,
price DOUBLE(16,2) DEFAULT '0.00' NOT NULL,
PRIMARY KEY(article, dealer));

INSERT INTO shop VALUES
(1,'A',3.45),(1,'B',3.99),(2,'A',10.99),
   (3,'B',1.45),(3,'C',1.69),(3,'D',1.25),
   (4,'D',19.95);
```

Exercise 1

- Find out the highest item article

```
SELECT MAX(article) AS article FROM shop;
```

- Find the article, dealer, and price of the most expensive article.

```
SELECT article, dealer, price
FROM shop
WHERE price=(SELECT MAX(price) FROM shop);

SELECT article, dealer, price
FROM shop
ORDER BY price DESC
LIMIT 1;
```

Exercise 1

- Find the highest price per article.

```
SELECT article, MAX(price) AS price
FROM shop
GROUP BY article;
```

- For each article, find the dealer(s)with the most expensive price.

```
SELECT article, dealer, price
FROM shop s1
WHERE price=(SELECT MAX(s2.price)
           FROM shop s2
           WHERE s1.article = s2.article);
```

Exercise 2

See Sec. 3.6.9 of the tutorial for more details

- Create the following tables

```
CREATE TABLE person (
id SMALLINT UNSIGNED NOT NULL AUTO_INCREMENT,
name VARCHAR(60) NOT NULL,
PRIMARY KEY (id) );

CREATE TABLE shirt (
id SMALLINT UNSIGNED NOT NULL AUTO_INCREMENT,
style ENUM('t-shirt', 'polo', 'dress') NOT NULL,
color ENUM('red', 'blue', 'orange', 'white',
   'black') NOT NULL,
owner SMALLINT UNSIGNED NOT NULL REFERENCES
   person(id),
PRIMARY KEY (id) );
```

Exercise 2

Populate the tables

```
INSERT INTO person VALUES (NULL, 'Antonio Paz');
SELECT @last := LAST_INSERT_ID();
INSERT INTO shirt VALUES
(NULL, 'polo', 'blue', @last),
(NULL, 'dress', 'white', @last),
(NULL, 't-shirt', 'blue', @last);

INSERT INTO person VALUES (NULL, 'Lilliana
   Angelovska');

SELECT @last := LAST_INSERT_ID();

INSERT INTO shirt VALUES
(NULL, 'dress', 'orange', @last),
(NULL, 'polo', 'red', @last),
(NULL, 'dress', 'blue', @last),
(NULL, 't-shirt', 'white', @last);
```

See Sec. 3.6.5, 3.6.9 of the tutorial for more details

Exercise 2

- Find out person names containing “Lilliana” as a string and having a shirt of any color but not white

```
SELECT s.*
FROM person p INNER JOIN shirt s ON s.owner =
   p.id
WHERE p.name LIKE '%Lilliana%' AND s.color <>
   'white';
```

Exercise 3: Banking Example

Create a DB "bank" having the following schema

- *branch (<u>branch-name</u>, branch-city, assets)*
- *customer (<u>customer-name</u>, customer-street, customer-city)*
- *account (<u>account-number</u>, branch-name, balance)*
- *loan (<u>loan-number</u>, branch-name, amount)*
- *depositor (<u>customer-name, account-number</u>)*
- *borrower (<u>customer-name, loan-number</u>)*
- *employee (<u>employee-name, branch-name</u>, salary)*

The SQL script for creating the tables of this DB can be found at

http://www.cs.kent.edu/~mabuata/DB10_lab/bank_db.sql

http://www.cs.kent.edu/~mabuata/DB10_lab/bank_data.sql

Suggestion: Use `source` command to execute a script file while staying in mySQL environment

Query

- Find out all loan number for loans made at the Perryridge branch with loan amounts greater than $1100.

```
select loan_number  from loan
where branch_name = 'Perryridge' and
amount>1100;
```

- Find out the loan number of those loans with loan amounts between $1,000 and $1,500 (that is, ≥$1,000 and ≤$1,500)

```
select loan_number from loan
where amount between 1000 and 1500;
```

Query

- Find the names of all branches that have greater assets than some branch located in Brooklyn.

```
select distinct T.branch_name
from branch as T, branch as S
where T.assets > S.assets and
S.branch_city = 'Brooklyn';
```

Try with ANY

```
select branch_name
from branch
where assets > ANY (SELECT assets from
  branch where branch_city = "Brooklyn");
```

Query

- Find the customer names, their loan numbers and loan amounts for all customers having a loan at some branch.

```
 select customer_name, T.loan_number,
S.amount
 from borrower as T, loan as S
 where  T.loan_number = S.loan_number;
```

Try with JOIN

Set Operation

- Find all customers who have a loan, an account, or both:

```
(select customer_name from depositor)
 union
(select customer_name from borrower);
```

- Find all customers who have an account but no loan (no **minus** operator provided in mysql)

```
select customer_name from depositor
where customer_name  not in
(select customer_name from borrower);
```

Aggregate function

- Find the number of depositor names for each branch

```
select branch_name, count(distinct customer_name)
from depositor, account
where depositor.account_number =
account.account_number
group by branch_name;
```

Try with join

- Find the names of all branches where the average account balance is more than $500

```
select branch_name, avg(balance)
from account
group by branch_name
having avg(balance) > 500;
```

Nested Subqueries

- Find all customers who have both an account and a loan at the bank (No intersect oper. available)

```
select distinct customer_name
from borrower
where customer_name in
   (select customer_name from depositor);
```

- Find all customers who have a loan at the bank but do not have an account at the bank

```
select distinct customer_name
from borrower
where customer_name not in
   (select customer_name from depositor);
```

Nested Subquery

- Find the names of all branches that have greater assets than all branches located in Horseneck

```
select branch_name
   from branch
   where assets > all
      (select assets
      from branch
      where branch_city = 'Horseneck');
```

Create View

(new feature in mysql 5.0)

A view consisting of branches and their customers

```
create view all_customer as
 (select branch_name, customer_name
 from depositor, account
 where depositor.account_number =
account.account_number)
  union
(select branch_name, customer_name
from borrower, loan
where borrower.loan_number=loan.loan_number);
```

Modification of Database

Increase all accounts with balances over $800 by 7%, all other accounts receive 8%.

```
update account
set balance = balance * 1.07
where balance > 800;

update account
set balance = balance *  1.08
where balance <= 800;
```

Modification of Database

Delete the record of all accounts with balances below the average at the bank.

```
select @avarageBalance := avg (balance) from account;

delete from account
 where balance < @avarageBalance;
```

Add a new tuple to *account*

```
insert into account
values ('A-9732', 'Perryridge',1200);
```

Exercise 4

- Build the DB having the following tables

Employees

<u>Code</u>	Name	Age	Salary
101	Mario Rossi	34	4000
103	Mario Bianchi	23	3500
104	Luigi Neri	38	6100
105	Nico Bini	44	3800
210	Marco Celli	49	6000
231	Siro Bisi	50	6000
252	Nico Bini	44	7000
301	Sergio Rossi	34	7000
375	Mario Rossi	50	6500

Supervision

Head	<u>Employee</u>
210	101
210	103
210	104
231	105
301	210
301	231
375	252

Exercise 4

- Create a new database

```
CREATE DATABASE employeesSupervision;
```

- Create Tables

```
CREATE TABLE employees (name VARCHAR(20) NOT
   NULL, surname VARCHAR(20) NOT NULL, age
   smallint UNSIGNED,  salary double(7,2), code
   smallint UNSIGNED NOT NULL AUTO_INCREMENT
   PRIMARY KEY );


CREATE TABLE supervision (employee smallint(5)
   unsigned primary key references
   employees(code), head smallint(5) unsigned
   references employees(code) );
```

Exercise 4

Populating tables (without auto_increment)

```
INSERT INTO employees VALUES ('mario', 'rossi', 34,
   4000, 101), ('mario', 'bianchi', 23, 3500, 103),
   ('luigi', 'neri', 38, 6100, 104), ('nico', 'bini',
   44, 3800, 105), ('marco', 'celli', 49, 6000, 210),
   ('Siro', 'Bisi', 50, 6000, 231), ('Nico', 'Bini',
   44, 7000, 252), ('Sergio', 'Rossi', 34, 7000, 301),
   ('Mario', 'Rossi', 50, 6500, 375);
```

```
INSERT INTO supervision VALUES (101, 210), (103, 210),
   (104, 210), (105, 231), (210, 301), (231, 301),
   (252, 375);
```

Exercise 4: Queries

***Q:** find code, name, surname, age and salary of the employees earning more than 4000 Euros*

```
select * from amployees where salary > 4000;
```

Tuple Relational Calculus:

$\{\ e.* \mid e(\text{Employees}) \mid e.\text{salary} > 4000\}$

Exercise 4: Queries

***Q:** find code, name, surname and age of the employees earning more than 4000 Euros*

```
select code, name, surname, age
from employees
where salary > 4000;
```

Tuple Relational Calculus:

{ e.(code,name,surname, age)| *e*(Employees) | e.salary >4000}

Exercise 4: Queries

Q: *find the codes of the heads of the employees which earn more than 4000 Euros*

```
select distinct head
from supervision s join employees e on
  e.code=s.employee
where salary > 4000;
```

Tuple Relational Calculus:

$\{ s.\text{head} \mid e(\text{Employees}), s(\text{Supervision}) \mid e.\text{code}= s.\text{employee} \wedge e.\text{salary}>4000\}$

Exercise 4: Queries

Q: *find name, surname and salary of the heads of the employees which earn more than 4000 Euros*

```
select distinct e2.name, e2.surname,
  e2.salary
from employees e1 join supervision s on
  e1.code=s.employee join employees e2 on
  s.head = e2.code
where e1.salary > 4000;
```

Tuple Relational Calculus:

{ NameH,SalaryH:e'.(name,salary)|
e'(Employees), *e*(Employees), *s*(Supervision) | *e*.code= *s*.employee ∧ *s*.head= *e*'.code ∧ *e*.salary>4000}

Exercise 4: Queries

Q: *find the employees which earn more than their respective heads. Show: code, name, surname and salary of such emplyees and their heads*

```
select distinct e1.code, e1.name, e1.surname,
    e1.salary, e2.code as headCode, e2.name as
    headName, e2.surname as headSurname, e2.salary
    as headSalary
from employees e1 join supervision s on
    e1.code=s.employee join employees e2 on s.head
    = e2.code
where e1.salary > e2.salary;
```

Tuple Relational Calculus:

{*e*.(name,code,salary), nameH, codeH, salarH:e'.(name, code, salary) | *e'*(Employees), *e*(Employees), *s*(Supervision) | *e*.code = *s*.employee ∧ *s*.head= *e'*.head ∧ *e*.Stipendio> *e'*.Stipendio }

Exercise 4: Queries

***Q:** find code, name and surname of the heads whose emploees all earn more than 4000 euros*

```
select distinct e.code, e.name, e.surname
from employees e join supervision s on
   s.head=e.code
where e.code not in (select head
                     from employees e1 join
                     supervision s1 on e1.code =
                        s1.employee
                     where e1.salary <= 4000);
```

Tuple Relational Calculus:

{e.(code, name) | e(Employees), s(Supervision) | e.code = s.head $\wedge$ $\neg$($\exists\epsilon$'(Employees) ($\exists s$'(Supervision) (s.head = s'.head $\wedge$ s'.employee = e'.code $\wedge$ e'.salary $\leq$ 4000)))}

Exercise 5

Given the DB schema:

FILM(<u>CodFilm</u>, Tit, CodDirector, Year, RentalCost)

ARTISTS(<u>CodActor</u>, Surname, Name, Sex, DateOfBirth, Nationality)

INTERNPRETATIONS(<u>CodFilm, CodActor, Celebrity</u>)

Write the SQL query for the following requirements

Q: find the titles of the movies where the director has been also interpreter/celebrity

Tuple Relational Calculus:

$\{F.Tit \mid F(FILM), I(INTERNPRETATIONS) \mid (F.CodFilm = I.CodFilm) \wedge (F.CodDirector = I.CodActor)\}$

Exercise 6

Given the DB schema:

COURSES(Cod, Faculty, CourseName, Prof)

STUDENTS(CodStud, Surname, Name, Faculty)

PROFESSORS(CodProf, Surname, Name)

EXAMS(Student, CodCourse, Evaluation, Date)

PLAN_OF_STUDY(Student, CodCourse, Year)

Write the SQL query for the following requirements

Q1: find the students that got an evaluation equal to 30 for at least one exam. For each student, show name, surname and the date of the first of such exams

Exercise 6

Tuple Relational Calculus:

{S.(Name,Surname), E.Date |
S(STUDENTS),E(EXAMS) | (S.CodStud=E.Student)
$\wedge$ (E.Evaluation=”30”) $\wedge$ $\neg$($\exists$ E1(EXAMS)
((E1.Student=S.CodStud) $\wedge$ (E1.Evaluation=”30”)
$\wedge$ (E1.Date < E.Date)))}

Exercise 6

Given the DB schema:

COURSES(Cod, Faculty, CourseName, Prof)

STUDENTS(CodStud, Surname, Name, Faculty)

PROFESSORS(CodProf, Surname, Name)

EXAMS(Student, CodCourse, Evaluation, Date)

PLAN_OF_STUDY(Student, CodCourse, Year)

Write the SQL query for the following requirements

Q2: find name and surname of the students that succeeded in *at least* one exam tought by a professor having the same student name

Exercise 6

{S.(Name, Surname) | S(STUDENTS), E(EXAMS), C(COURSES), P(PROFESSORS) | (S.CodStud = E.Student) ∧ (E.CodCourse=C.Cod) ∧ (C.Prof=P.CodProf) ∧ (P.Name=S.Name)}

Exercise

Given the following DB schema:

Students(<u>Code</u>, Name, Province, YearOfBirth)

Exams(<u>Course</u>, <u>StudentCode</u>, Evalaution, NumCall, Year)

Write the SQL queries for the following requests:

- Find the name of the students residing in Taranto and that succeeded with the Data Base exam with evaluation equals to 30
- Find the name of the studends that succeeded in 5 exams
- Find, for each student residing in Taranto, the number of succeeded exams, the highest, the lowest and the average evaluation
- Find, for each course, the number of exams succeeded at the first call and the highest, the lowest and the avarage evalaution