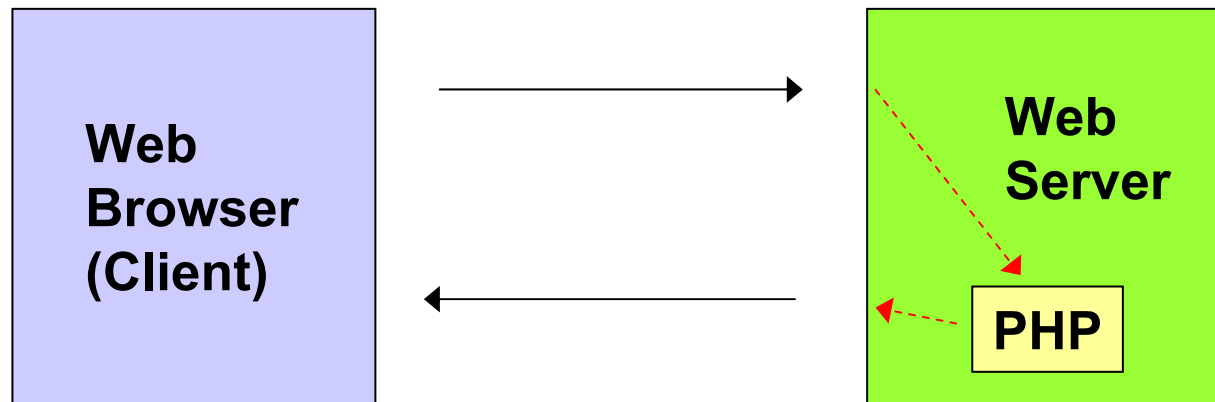




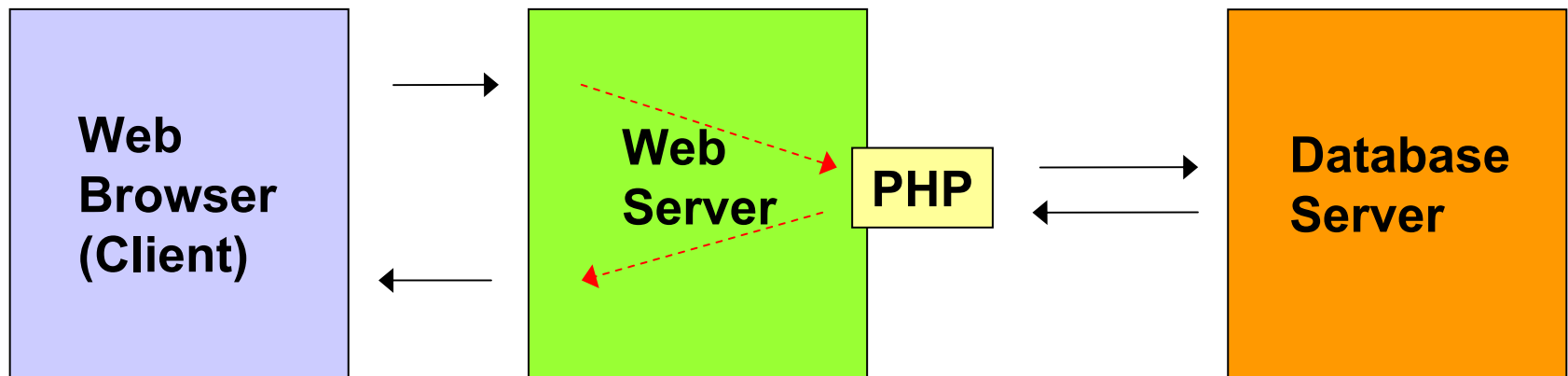
COSC 2206 Internet Tools

MySQL Database System
Installation Overview
SQL summary

2-Tier Architecture



3-Tier Architecture





SQL links

■ Tutorials

- <http://www.w3schools.com/sql/>
- <http://www.sqlzoo.net>
- <http://sqlcourse.com> (part 2)
- <http://sqlcourse2.com> (part 1)

■ MySQL online reference manual

- <http://dev.mysql.com/doc/mysql/en/Reference.html>



Installation Summary

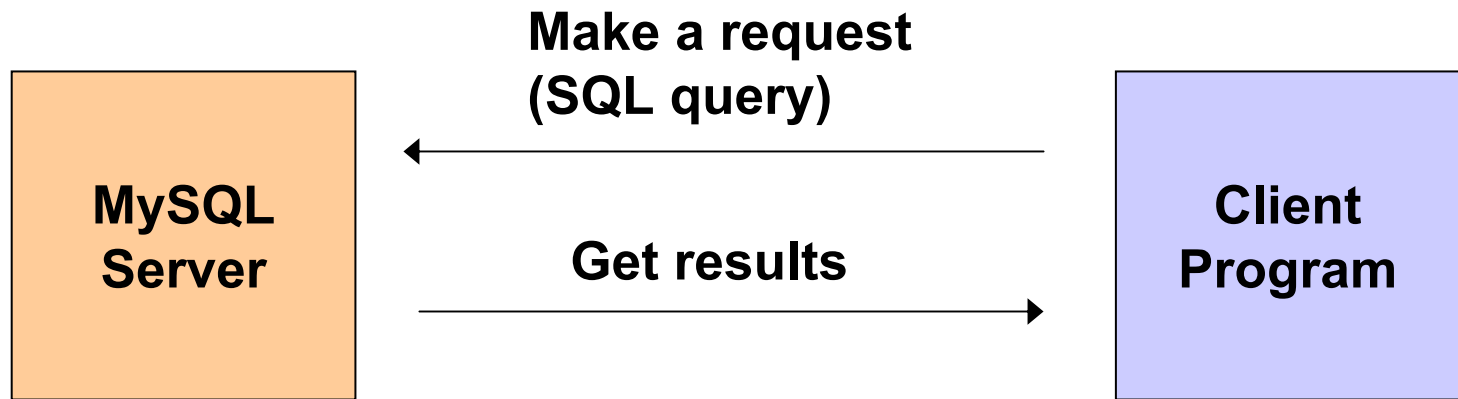
- More detailed installation instructions are given on the CD
- Install MySQL in `c:\mysql`
- MySQL can be installed as a service (Win 2000/XP)
- Can make icons on the desktop for starting and stopping the server.



Command Line Client

- The standard command line client is
- `c:\mysql\bin\mysql.exe`
- The command line client can be used to send commands and SQL queries to the MySQL server
- There are also GUI clients such as MyCC

Client-Server Interaction



Client program can be a MySQL command line client, GUI client, or a program written in any language such as C, Perl, PHP, Java that has an interface to the MySQL server.



Connecting to the Server

- Use a command prompt that sets the path to `c:\mysql\bin`
- The following command connects to the server:
 - `mysql -u root -p`
 - you are prompted for the root password.
 - you can now send commands and SQL statements to the server



WARNING WARNING

■ WARNING

- Always assume that everything is case sensitive, especially table names.
- This is not the case in Windows XP but it is the case in Linux



Entering commands (1)

- Show all the databases
 - **SHOW DATABASES;**

```
mysql> SHOW DATABASES;  
+-----+  
| Database |  
+-----+  
| bookstore |  
| employee_db |  
| mysql |  
| student_db |  
| test |  
| web_db |  
+-----+
```



Entering commands (2)

- Choosing a database and showing its tables
 - **USE test;**
SHOW tables;

```
mysql> USE test;
Database changed
mysql> SHOW tables;
+-----+
| Tables_in_test |
+-----+
| books          |
| name2          |
| names          |
| test           |
+-----+
4 rows in set (0.00 sec)
mysql>
```

Entering commands (3)

- Show the structure of a table
 - **DESCRIBE names;**

```
mysql> DESCRIBE names;
```

Field	Type	Null	Key	Default	Extra
id	int(11)		PRI	NULL	auto_increment
firstName	varchar(20)				
lastName	varchar(20)				

```
3 rows in set (0.00 sec)
```

```
mysql>
```

Entering commands (4)

- Show the rows of a table (all columns)
 - **SELECT * FROM names;**

```
mysql> SELECT * FROM names;
+----+-----+-----+
| id | firstName | lastName |
+----+-----+-----+
|  1 | Fred      | Flintstone |
|  2 | Barney    | Rubble     |
+----+-----+-----+
2 rows in set (0.00 sec)

mysql>
```

Entering commands (5)

■ Inserting a new record

- `INSERT INTO names (firstName, lastName) VALUES ('Rock', 'Quarry');`
- `SELECT * FROM names;`

```
mysql> INSERT INTO names (firstName, lastName) VALUES ('Ralph', 'Quarry');
Query OK, 1 row affected (0.02 sec)
mysql> SELECT * FROM names;
+----+-----+-----+
| id | firstName | lastName |
+----+-----+-----+
| 1  | Fred      | Flintstone |
| 2  | Barney    | Rubble      |
| 3  | Ralph     | Quarry      |
+----+-----+-----+
3 rows in set (0.00 sec)
mysql>
```

Entering commands (6)

■ Updating a record

- **UPDATE** names SET lastName = 'Stone' WHERE id=3;
- **SELECT * FROM** names;

```
mysql> UPDATE names SET lastName = 'Stone' WHERE id=3;
Query OK, 1 row affected (0.28 sec)
Rows matched: 1  Changed: 1  Warnings: 0
mysql> SELECT * FROM names;
+----+-----+-----+
| id | firstName | lastName |
+----+-----+-----+
| 1  | Fred      | Flintstone |
| 2  | Barney    | Rubble     |
| 3  | Ralph     | Stone      |
+----+-----+-----+
3 rows in set (0.00 sec)
mysql>
```



Logging output

- The commands you type and their output can be logged to a file by using the following command inside the MySQL command line client
- `tee log.txt`
- Here log.txt is the name of the file



Executing SQL files (1)

- It is usually better to use an editor to write an SQL script and send it to the server.
- A file of SQL commands such as **books.sql** can be executed by the server by using a command such as
 - `C:\mysql\bin\mysql -u root -p < books.sql`
- This assumes that **books.sql** is in your current directory. Otherwise the complete path to **books.sql** must be supplied



Executing SQL files (2)

- A file of SQL commands such as **books.sql** can also be executed from inside the MySQL client using the **source** command
 - `source c:\...\books.sql`
- Here the full path to **books.sql** should be used.



Documentation

- MySQL comes with a tutorial and complete documentation in a HUGE file:
 - `c:\mysql\Docs\manual.html`
- Table of contents with links:
 - `c:\mysql\Docs\manual_toc.html`
- Use this file to locate the link to the topic you are interested in.

Database concepts (1)

- A relational database management system consists of a number of databases.
- Each database consists of a number of tables.
- Example table

books table

isbn	title	author	pub	year	price

column headings

rows (records)



Some SQL data types (1)

- Each entry in a row has a type specified by the column.
- Numeric data types
 - `TINYINT`, `SMALLINT`, `MEDIUMINT`,
 - `INT`, `BIGINT`
 - `FLOAT(display_length, decimals)`
 - `DOUBLE(display_length, decimals)`
 - `DECIMAL(display_length, decimals)`
 - `NUMERIC` is the same as `DECIMAL`



Some SQL data types (2)

- Date and time types
 - **DATE**
 - format is YYYY-MM-DD
 - **DATETIME**
 - format YYYY-MM-DD HH:MM:SS
 - **TIMESTAMP**
 - format YYYYMMDDHHMMSS
 - **TIME**
 - format HH:MM:SS
 - **YEAR**
 - default length is 4



SQL data types (3)

■ String types

■ **CHAR**

- fixed length string, e.g., **CHAR(20)**

■ **VARCHAR**

- variable length string, e.g., **VARCHAR(20)**

■ **BLOB, TINYBLOB, MEDIUMBLOB, LONGBLOB**

- same as **TEXT, TINYTEXT ...**

■ **ENUM**

- list of items from which value is selected



SQL commands SHOW, USE

■ SHOW

- Display databases or tables in current database;
- Example (command line client):
 - `show databases;`
 - `show tables;`

■ USE

- Specify which database to use
- Example
 - `use bookstore;`



The CREATE Command (1)

- **CREATE** creates a database table

```
CREATE TABLE table_name
(
    column_name1 column_type1,
    column_name2 column_type2,
    ...
    column_nameN column_typeN
);
```

Note: To create a database use the statement

```
CREATE db_name;
```



The CREATE Command (2)

- Specifying primary keys

```
CREATE TABLE table_name
(
    column_name1 column_type1 NOT NULL
        DEFAULT '0',
    column_name2 column_type2,
    ...
    column_nameN column_typeN,
    PRIMARY KEY (column_name1)
);
```



The CREATE Command (3)

- autoincrement primary integer keys

```
CREATE TABLE table_name
(
    column_name1 column_type1 PRIMARY
    KEY NOT NULL DEFAULT '0'
    AUTO_INCREMENT,
    column_name2 column_type2,
    ...
    column_nameN column_typeN,
);
```



The CREATE Command (4)

- Can also create UNIQUE keys. They are similar to PRIMARY KEYS but can have NULL values.
- Can also create INDEX fields.



Conditional Creation

- Conditional database creation
 - **CREATE DATABASE IF NOT EXISTS**
db_name;
- Conditional table creation
 - **CREATE TABLE IF NOT EXISTS**
table_name;



The DROP Command

- To delete databases and tables use the DROP command
- Examples
 - DROP DATABASE db_name;
 - DROP DATABASE IF EXISTS db_name;
 - DROP TABLE table_name;
 - DROP TABLE IF EXISTS table_name;

Note: Don't confuse DROP with DELETE which deletes rows of a table.



The INSERT Command

- Inserting rows into a table

```
INSERT INTO table_name  
    ( col_1, col_2, ..., col_N)  
VALUES  
    ( val_1, val_2, ..., val_N) ;
```

String values are enclosed in single quotes by default but double quotes are also allowed. Literal quotes need to be escaped using \' and \"



The SELECT Command (1)

- Selecting rows from a table
- Simplest form: select all columns

```
SELECT * FROM table_name;
```

- Select specified columns

```
SELECT column_list FROM table_name;
```

- Conditional selection of rows

```
SELECT column_list FROM table_name  
WHERE condition;
```




The SELECT Command (2)

- Specifying ascending row ordering

```
SELECT column_list FROM table_name  
WHERE condition  
ORDER by ASC;
```

- Specifying descending row ordering

```
SELECT column_list FROM table_name  
WHERE condition  
ORDER by DESC;
```



The SELECT Command (3)

- There are many other variations of the select command.
- Example: finding the number of records in a table assuming a primary key called `id`:

```
SELECT COUNT(id) FROM table_name
```

- Can also perform searching using the `WHERE` option



The UPDATE Command

- Used to modify an existing record

```
UPDATE table_name  
SET col_1 = 'new_value1',  
..., col_n = 'new_value2';
```

- Conditional update version

```
UPDATE table_name  
SET col_1 = 'new_value1',  
..., col_n = 'new_value2'  
WHERE condition;
```



marks.sql (1)

marks
table

studentID	first_name	last_name	mark

```
USE test;
CREATE TABLE marks (
  studentID SMALLINT AUTO_INCREMENT NOT NULL,
  first_name VARCHAR(20) NOT NULL,
  last_name VARCHAR(20) NOT NULL,
  mark SMALLINT DEFAULT 0 NOT NULL,
  PRIMARY KEY (studentID)
);
```



marks.sql (2)

```
-- Insert some rows into marks table
```

```
INSERT INTO marks (first_name, last_name,  
mark) VALUES ('Fred', 'Jones', 78);
```

```
INSERT INTO marks (first_name, last_name,  
mark) VALUES ('Bill', 'James', 67);
```

```
INSERT INTO marks (first_name, last_name,  
mark) VALUES ('Carol', 'Smith', 82);
```

```
INSERT INTO marks (first_name, last_name,  
mark) VALUES ('Bob', 'Duncan', 60);
```

```
INSERT INTO marks (first_name, last_name,  
mark) VALUES ('Joan', 'Davis', 86);
```



Executing The Script

- within MySQL use a command such as
 - **source c:/...../marks.sql**
- This adds the **marks** table to the **test** database



The Marks Table

- Selecting the complete table

```
SELECT * FROM marks;
```

```
+-----+-----+-----+-----+
| studentID | first_name | last_name | mark |
+-----+-----+-----+-----+
|          1 | Fred      | Jones     | 78   |
|          2 | Bill      | James     | 67   |
|          3 | Carol     | Smith     | 82   |
|          4 | Bob       | Duncan    | 60   |
|          5 | Joan      | Davis     | 86   |
+-----+-----+-----+-----+
5 rows in set (0.00 sec)
```



The WHERE Clause (1)

- Select rows according to some criterion

```
SELECT * FROM marks WHERE studentID > 1
      AND studentID < 5;
```

```
+-----+-----+-----+-----+
| studentID | first_name | last_name | mark |
+-----+-----+-----+-----+
|          2 | Bill      | James    | 67   |
|          3 | Carol     | Smith    | 82   |
|          4 | Bob       | Duncan   | 60   |
+-----+-----+-----+-----+
3 rows in set (0.01 sec)
```


The WHERE Clause (2)

- Select rows with marks ≥ 80

```
SELECT * FROM marks WHERE mark >= 80;
```

```
+-----+-----+-----+-----+
| studentID | first_name | last_name | mark |
+-----+-----+-----+-----+
|          3 | Carol      | Smith     | 82   |
|          5 | Joan       | Davis     | 86   |
+-----+-----+-----+-----+
2 rows in set (0.00 sec)
```

The ORDER BY Clause

- Select rows according to some criterion

```
SELECT * FROM marks ORDER BY mark DESC;
```

studentID	first_name	last_name	mark
5	Joan	Davis	86
3	Carol	Smith	82
1	Fred	Jones	78
2	Bill	James	67
4	Bob	Duncan	60

5 rows in set (0.00 sec)



Searching Using LIKE (1)

- LIKE is used to search a table for values containing a search string:
- There are two wild-card characters used to specify patterns:
 - _ matches a single character
 - % matches zero or more characters
- Can also use NOT LIKE
- Searching is case insensitive



Searching Using LIKE (2)

- Example: last names in marks table that begin with J

```
SELECT * FROM marks WHERE last_name  
LIKE 'J%';
```

- Example: first names that have 3 letters

```
SELECT * FROM marks WHERE first_name  
LIKE '___';
```



Quoting strings

- If a string contains a single quote it must be backquoted (escaped) before it can be used in a query
- Example: find records containing O'Reilly in the `last_name` field.

```
SELECT * FROM marks WHERE last_name  
= 'O\'Reilly';
```



Limiting number of rows

- **LIMIT** can be used to specify the maximum number of rows that are to be returned by a select query. Example
 - **SELECT * FROM marks LIMIT 3;**
- This query will return only the first 3 rows from the **marks** table
- To return 15 rows beginning at row 5 use
 - **SELECT * FROM marks LIMIT 4, 15;**



MySQL Functions (1)

- How many rows are there ?

```
SELECT COUNT (*) FROM marks;
```

```
+-----+
| COUNT (*) |
+-----+
|          5 |
+-----+
1 row in set (0.00 sec)
```

- Can use **COUNT** (marks) instead of **COUNT** (*)



MySQL Functions (2)

- What is the sum of all the marks?

```
SELECT SUM(mark) FROM marks;
```

```
+-----+  
| SUM(mark) |  
+-----+  
|          373 |  
+-----+  
1 row in set (0.00 sec)
```




MySQL Functions (3)

- What is the average mark?

```
SELECT AVG(mark) FROM marks;
```

```
+-----+  
| AVG(mark) |  
+-----+  
|    74.6000 |  
+-----+  
1 row in set (0.00 sec)
```



MySQL Functions (4)

- What is the minimum mark?

```
SELECT MIN(mark) FROM marks;
```

```
+-----+  
| MIN(mark) |  
+-----+  
|          60 |  
+-----+  
1 row in set (0.00 sec)
```



MySQL Functions (5)

- What is the maximum mark?

```
SELECT MAX(mark) FROM marks;
```

```
+-----+  
| MAX(mark) |  
+-----+  
|          86 |  
+-----+  
1 row in set (0.00 sec)
```

books.sql (1)

books
table

isbn	title	author	pub	year	price

this is a
simple
design

```
USE web_db;  
CREATE TABLE books (  
    isbn CHAR(15) PRIMARY KEY NOT NULL,  
    title VARCHAR(100) NOT NULL,  
    author VARCHAR(100) NOT NULL,  
    pub VARCHAR(20) NOT NULL,  
    year YEAR NOT NULL,  
    price DECIMAL(9,2) DEFAULT NULL  
);
```



books.sql (2)

```
-- Insert some books into books table
```

```
INSERT INTO books VALUES ('0-672-31784-2',  
    'PHP and MySQL Web Development',  
    'Luke Welling, Laura Thomson',  
    'Sams', 2001, 74.95  
);
```

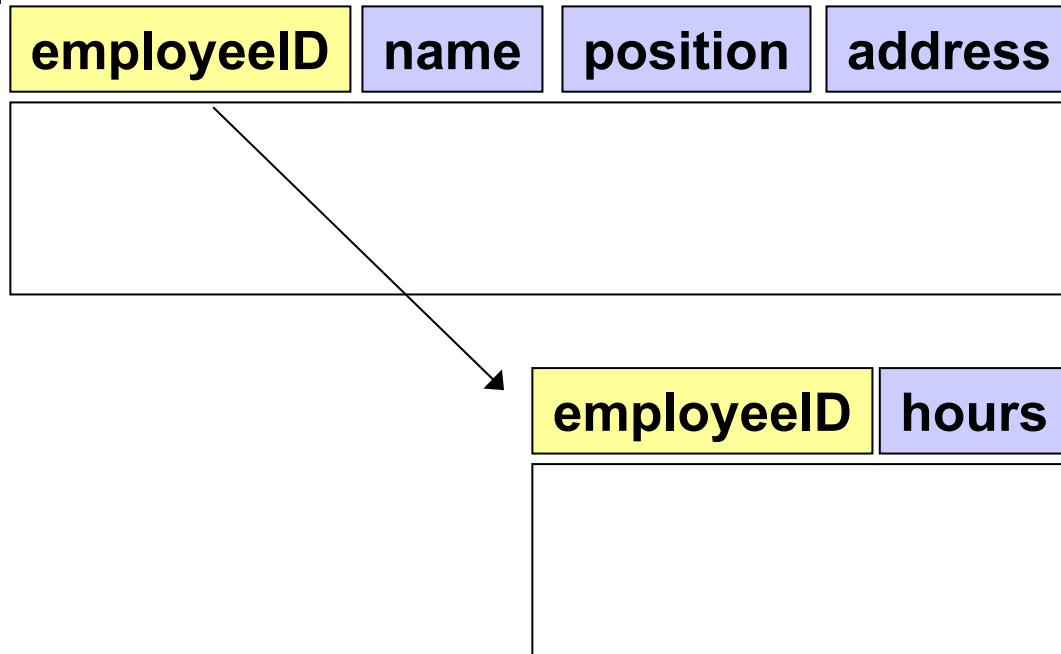
```
INSERT INTO books VALUES ('1-861003-02-1',  
    'Professional Apache',  
    'Peter Wainwright',  
    'Wrox Press Ltd', 1999, 74.95  
);
```



Executing The Script

- within MySQL use a command such as
 - **source c:/...../books.sql**
- This adds the books table to the **web_db** database

employee_db.sql (1)



**employees
table**

**jobs
table**

```
CREATE DATABASE IF NOT EXISTS employee_db;  
USE employee_db;  
DROP TABLE IF EXISTS employees;  
DROP TABLE IF EXISTS jobs;
```



employee_db.sql (1)

```
CREATE TABLE employees (  
    employeeID SMALLINT NOT NULL,  
    name VARCHAR(20) NOT NULL,  
    position VARCHAR(20) NOT NULL,  
    address VARCHAR(40) NOT NULL,  
    PRIMARY KEY (employeeID)  
);  
  
INSERT INTO employees VALUES (1001, 'Fred',  
    'programmer', '13 Windle St');  
INSERT INTO employees VALUES (1002, 'Joan',  
    'programmer', '23 Rock St');  
INSERT INTO employees VALUES (1003, 'Bill',  
    'manager', '37 Front St');
```




employee_db.sql (2)

```
CREATE TABLE jobs (  
    employeeID SMALLINT NOT NULL,  
    hours DECIMAL(5,2) NOT NULL,  
);  
  
INSERT INTO jobs VALUES (1001, 13.5);  
INSERT INTO jobs VALUES (1002, 2);  
INSERT INTO jobs VALUES (1002, 6.25);  
INSERT INTO jobs VALUES (1003, 4);  
INSERT INTO jobs VALUES (1001, 1);  
INSERT INTO jobs VALUES (1003, 7);  
INSERT INTO jobs VALUES (1003, 9.5);
```



Executing The Script

- within MySQL use a command such as
 - `source c:/...../employee_db.sql`
- This creates the `employee_db` database and adds the `employees` and `jobs` tables to it

Select Queries With Joins (1)

■ Cartesian product query

```
SELECT * FROM employees, jobs;
```

employeeID	name	position	address	employeeID	hours
1001	Fred	programmer	13 Windle St	1001	13.50
1002	Joan	programmer	23 Rock St	1001	13.50
1003	Bill	manager	37 Front St	1001	13.50
1001	Fred	programmer	13 Windle St	1002	2.00
1002	Joan	programmer	23 Rock St	1002	2.00
1003	Bill	manager	37 Front St	1002	2.00
1001	Fred	programmer	13 Windle St	1002	6.25
1002	Joan	programmer	23 Rock St	1002	6.25
1003	Bill	manager	37 Front St	1002	6.25

Select Queries With Joins (2)

■ Cartesian product query (continued)

	1001		Fred		programmer		13 Windle St		1003		4.00	
	1002		Joan		programmer		23 Rock St		1003		4.00	
	1003		Bill		manager		37 Front St		1003		4.00	
	1001		Fred		programmer		13 Windle St		1001		1.00	
	1002		Joan		programmer		23 Rock St		1001		1.00	
	1003		Bill		manager		37 Front St		1001		1.00	
	1001		Fred		programmer		13 Windle St		1003		7.00	
	1002		Joan		programmer		23 Rock St		1003		7.00	
	1003		Bill		manager		37 Front St		1003		7.00	
	1001		Fred		programmer		13 Windle St		1003		9.50	
	1002		Joan		programmer		23 Rock St		1003		9.50	
	1003		Bill		manager		37 Front St		1003		9.50	
+-----+-----+-----+-----+-----+-----+												
21 rows in set (0.01 sec)												

The cartesian product query is rarely what we want.

Select Queries With Joins (3)

■ Substitution

```
SELECT name, hours FROM employees, jobs WHERE  
employees.employeeID = jobs.employeeID;
```

```
+-----+-----+  
| name | hours |  
+-----+-----+  
| Fred | 13.50 |  
| Joan | 2.00  |  
| Joan | 6.25  |  
| Bill | 4.00  |  
| Fred | 1.00  |  
| Bill | 7.00  |  
| Bill | 9.50  |  
+-----+-----+
```

7 rows in set (0.00 sec)

Here we are replacing
the employeeID
numbers in the jobs
table by the employee's
name



Select Queries With Joins (4)

■ Entries only for Fred

```
SELECT name, hours FROM employees, jobs WHERE  
employees.employeeID = jobs.employeeID AND  
name = 'Fred';
```

```
+-----+-----+  
| name | hours |  
+-----+-----+  
| Fred | 13.50 |  
| Fred |  1.00 |  
+-----+-----+  
  
2 rows in set (0.00 sec)
```



Select Queries With Joins (5)

- Total hours worked for each person

```
SELECT name, SUM(hours) FROM employees, jobs
WHERE employees.employeeID = jobs.employeeID
GROUP BY name;
```

```
+-----+-----+
| name | SUM(hours) |
+-----+-----+
| Bill |      20.50 |
| Fred |      14.50 |
| Joan |       8.25 |
+-----+-----+
3 rows in set (0.00 sec)
```



Select Queries With Joins (6)

- Total hours worked, for Fred

```
SELECT name, SUM(hours) FROM employees, jobs
WHERE employees.employeeID = jobs.employeeID
AND name = 'Fred' GROUP BY name;
```

```
+-----+-----+
| name | SUM(hours) |
+-----+-----+
| Fred |      14.50 |
+-----+-----+
1 row in set (0.00 sec)
```