# CSCU9B3

# MySQL 2

# Data Manipulation

# Getting Data From a Database with MySQL

- The SQL **`SELECT`** statement is used to retrieve data from one or more tables in a database
- It allows you to choose which row and which columns you would like to retrieve and allows joins across several tables.

http://dev.mysql.com/doc/refman/5.7/en/select.html

# Example Tables for This Lecture

- Books

| *Name* | ***Number*** |
|---|---|
| Book1 | 1 |
| Book2 | 2 |
| Book3 | 3 |
| Book4 | 4 |
| Book5 | 5 |

Borrowers

| *Name* | ***Number*** | *Dept* |
|---|---|---|
| Anne | 1 | Maths |
| Bill | 2 | Maths |
| Claire | 3 | French |
| Duncan | 4 | French |
| Edward | 5 | French |

- Loans

| *BookNumber* | *PersonNumber* |
|---|---|
| 1 | 2 |
| 3 | 4 |

# SELECT From One Table

- **`SELECT * FROM Borrowers`**
  Shows whole table

- **`SELECT Name FROM Borrowers`**
  Lists Borrowers names only

- **`SELECT Number, Dept FROM Borrowers WHERE Name="Anne"`**
  Get Anne's number and department

# SELECT and Calculations

- There are many functions that you can include in a SELECT statement, for example:
  - **`SELECT MAX(Number) FROM Borrowers`**
    Shows largest borrower number (Min is also available)
  - **`SELECT Number+1 FROM Borrowers`**
    Adds 1 to each borrower number and reports it
  - **`SELECT sin(45)`**
    Calculates the sine of 45 – no need to reference a table at all

# Joining Tables

```
SELECT Borrowers.Name,
Books.Name FROM Borrowers, Books
```

- Pairs every borrower with every book
- Not an awful lot of use
- Needs to be expanded to be useful …

| Name | Name |
|---|---|
| Anne | Book1 |
| Bill | Book1 |
| Claire | Book1 |
| Duncan | Book1 |
| Edward | Book1 |
| Anne | Book2 |
| Bill | Book2 |
| Claire | Book2 |
| Duncan | Book2 |
| Edward | Book2 |
| Anne | Book3 |
| Bill | Book3 |
| Claire | Book3 |
| Duncan | Book3 |
| Edward | Book3 |
| Anne | Book4 |
| Bill | Book4 |
| Claire | Book4 |
| Duncan | Book4 |
| Edward | Book4 |
| Anne | Book5 |
| Bill | Book5 |
| Claire | Book5 |
| Duncan | Book5 |
| Edward | Book5 |

# Looking up a Foreign Key

- Let's say we want to know who has borrowed Book1
- The Loans table has this information, but not in a form that is immediately accessible:

| *BookNumber* | *PersonNumber* |
|---|---|
| 1 | 2 |
| 3 | 4 |

- **`SELECT Number FROM Books WHERE Name="Book1"`**
  Tells us that book1 has the ID number 1
- **`SELECT PersonNumber FROM Loans WHERE BookNumber=1`**
  Tells us that person 2 has the book
- **`SELECT Name FROM Borrowers WHERE Number=2`**
  Tells us that person number 2 is Bill, so Bill has Book1

# Put it all Together

```
SELECT Borrowers.Name FROM Borrowers, Books, Loans
WHERE Books.Name="Book1" AND
Books.Number=Loans.BookNumber
AND Borrowers.Number=Loans.PersonNumber
```

- Returns Bill.
- Note the use of **AND**.
- You can also use **OR** and **NOT**.
- And comparisons: **>**, **<**, **<=**, **>=**, **=**, **!=** (or **<>**)

# Other Useful Logic

- When comparing NULL, use **`IS NULL`** or **`IS NOT NULL`**

- You can search for partial strings using **`LIKE`**:

```
SELECT Name FROM Borrowers WHERE Name LIKE "%e"
```

The `%` character is a wild card, so above looks for anything ending in e

```
SELECT Name FROM Borrowers WHERE Name LIKE "_e"
```

The `_` is a single character wild card, so above looks for names with two characters ending in e

# Comparison Examples

```
SELECT * FROM table WHERE x BETWEEN 1 and 3
SELECT * FROM table WHERE x IN ('A','B','C')
SELECT * FROM table WHERE x NOT IN ('A','B','C')
```

- Use brackets to enforce operator order:

```
WHERE (a=1) AND (b=2 OR b=3)
```

!=

```
WHERE (a=1 AND b=2) OR (b=3)
```

# SQL and Case Sensitivity

- Commands and Names (such as table or field names) are not case sensitive:
  - Select = SELECT
  - Table = table
  - To be sure, be consistent in your usage!
- String literals may be case sensitive, depending on the collation.
  - _ci at the end of the collation name means case insensitive
  - Force a case sensitive search with:

```
SELECT * FROM table WHERE BINARY name="John"
```

http://dev.mysql.com/doc/refman/5.0/en/identifier-case-sensitivity.html

# How to Construct a SELECT

- Decide what columns you want in your results table:

```
SELECT col1, col2 …
```

- Then list all the tables involved in the selection (some will not actually provide a field in the select list, but are involved anyway

```
SELECT col1, col2 FROM table1, table2, table3
```

- Then follow the path from what you want to look up to the answer, building **x=y** statements joined by **AND** (or **OR**).

```
SELECT col1, col2 FROM table1, table2, table3
WHERE table1.col1="target" AND
table2.col1=table1.col1 …
```

- This is known as performing a Natural Join

# More General Joins

- If we want to list all the people who have a book, and the books they have:

```
SELECT Borrowers.Name, Books.Name FROM Borrowers,
Books, Loans WHERE Books.Number=Loans.BookNumber
AND Borrowers.Number=Loans.PersonNumber
```

- Note that

```
SELECT * FROM Borrowers, Books, Loans WHERE
Books.Number=Loans.BookNumber AND
Borrowers.Number=Loans.PersonNumber
```

  - Is an Equijoin, and lists all the columns from all tables that meet the selection criteria, including them once for each table they appear in. This is not very useful, but shows that everything matches up.

# Inner and Outer Joins

- We have just seen an inner join. Only rows that meet the criteria completely are returned.
- Now we look at outer joins – where more data is returned about one or more of the fields.
- Let us say we want to list all of the people in the borrowers database, regardless of whether or not they have a book out. If they do have a book, we want to list that too, otherwise we will just return NULL for people with no current book
- For this, we need the **`JOIN`** command:

http://dev.mysql.com/doc/refman/5.7/en/join.html

# Join Syntax

```
SELECT Borrowers.Name, Books.Name FROM Borrowers LEFT
OUTER JOIN (Books, Loans) ON
(Books.Number=Loans.BookNumber AND
Borrowers.Number=Loans.PersonNumber)
```

| Name | Name |
|---|---|
| Anne | *NULL* |
| Bill | Book1 |
| Claire | *NULL* |
| Duncan | Book3 |
| Edward | *NULL* |

- Note the syntax

```
SELECT cols FROM table1 LEFT OUTER JOIN
(table2,table3) ON (conditions)
```

- You can use similar syntax for an inner join:

```
SELECT Borrowers.Name, Books.Name FROM Borrowers INNER
JOIN (Books, Loans) ON (Books.Number=Loans.BookNumber
AND Borrowers.Number=Loans.PersonNumber)
```

| name | name |
|---|---|
| Bill | Book1 |
| Duncan | Book3 |

  - Is the same as previous example of a general join

# A Contrived Example

- Who has the book with the lowest ID number?

```
SELECT Borrowers.Name FROM Borrowers, Books, Loans
WHERE Books.Number = (SELECT MIN(Number) FROM Books)
AND Books.Number=Loans.BookNumber
AND Borrowers.Number=Loans.PersonNumber
```

- The answer, as we would expect, is Bill
- Note the use of brackets to enclose the sub-SELECT and use of the function **`MIN()`**

# Groups

- We may want to summarise the data in the database, and there are some statistical functions available.
- We have already seen **MAX** and **MIN** and there are others:
  - **AVG**
  - **COUNT**

| Dept | COUNT( Dept ) |
|---|---|
| French | 3 |
| Maths | 2 |

- For example, how many people are there from each department?

```
SELECT Dept, COUNT(Dept) FROM Borrowers GROUP BY Dept
```

# How Many Books Does Each Department Have?

- We could show the list and count them ourselves:

```
SELECT Borrowers.Dept FROM Borrowers, Books, Loans
WHERE  Books.Number=Loans.BookNumber
AND Borrowers.Number=Loans.PersonNumber
```

| Dept | COUNT( Dept ) |
|---|---|
| French | 1 |
| Maths | 1 |

- But SQL can do it for us:

```
SELECT Dept, COUNT(Dept) FROM Borrowers, Books, Loans
WHERE  Books.Number=Loans.BookNumber
AND Borrowers.Number=Loans.PersonNumber GROUP BY Dept
```

- You can select from the resultant table using `HAVING`:

```
SELECT Dept, COUNT(Dept) FROM Borrowers, Books, Loans
WHERE  Books.Number=Loans.BookNumber
AND Borrowers.Number=Loans.PersonNumber GROUP BY Dept
HAVING COUNT(Dept) > 2
```

# Another Example

```
SELECT Dept, COUNT(Dept) FROM Borrowers, Books, Loans
WHERE  Books.Number=Loans.BookNumber
AND Borrowers.Number=Loans.PersonNumber GROUP BY Dept
HAVING COUNT(Dept) = (SELECT MAX(COUNT(Dept)))
```

- What does this return?

# Union

- Allows you to make more than one selection at the same time and put the results together
- A bit like using OR, but more flexible:

```
SELECT Name FROM Borrowers WHERE Name="Anne" OR
Name="Claire"
```

- Is the same as

```
SELECT Name FROM Borrowers WHERE Name="Anne"
UNION
SELECT Name FROM Borrowers WHERE Name="Claire"
```

- Which might seem pointless…

# Union

- But it is useful when selecting across more than one table

```
SELECT Name FROM Borrowers
UNION
SELECT Name FROM Staff
```

  - Selects the names of all borrowers and all staff
  - Note that there is an Anne in both tables. UNION will only list Anne once. To see duplicate rows, use

```
UNION ALL
```

**Staff**

| *Name* | ***Number*** | *Role* |
|---|---|---|
| Anne | 1 | Manager |
| Fred | 2 | Engineer |
| George | 3 | Engineer |
| Harry | 4 | Engineer |

# Selection Manipulation

Here are a few things you can do to the selected list:

- Sort the results by one or more fields

```
ORDER BY field, field, …  [DESC]
```

- Force a query to contain unique entries only:

```
SELECT DISTINCT …
```

- Select a given number of entries starting at a given offset

```
SELECT * FROM table LIMIT(offset,count)
```

# More on Sub Queries

We can use a sub query, but what happens when the sub query produces more than 1 row?

```
SELECT * FROM Staff WHERE Name = ANY (SELECT Name FROM Borrowers)
```

- Selects staff whose name appears in both Staff and Borrowers tables

```
SELECT * FROM Borrowers WHERE Number > ALL (SELECT Number FROM Staff)
```

- Selects those borrowers who have a higher number than ALL of the staff

# Aliases

- You can give an alias to a table or column to make it easier to refer to later in a query or to make it easier to read in the output:

```
SELECT AVG(Number) FROM books
```

AVG(Number)

3

```
SELECT AVG(Number) Average FROM books
```

Average

3

# Aliases (contd)

```
SELECT Role, COUNT(Role) FROM staff
GROUP BY Role HAVING COUNT(Role) > 1
```

- Can be re-written as:

```
SELECT Role, COUNT(Role) count FROM Staff
GROUP BY Role HAVING count > 1
```

  - Where count is an alias for COUNT(Role)

- You can also do this sort of thing:

```
SELECT * FROM Books a, Borrowers b WHERE
a.Number = b.Number
```

  - We have made aliases for the tables to make reference to them easier

# End of Lecture

- Next SQL lecture will look at SQL data types in more detail