**ADVANCED**

**SQL & MySQL**

**Performing joins**

Inner joins

A join is an SQL query that uses 2 or more tables, and produces a virtual table of results. Any time you need to simultaneously retrieve information from more than one table, a join is what you will probably use:

SELECT what\_columns FROM tableA

JOIN\_TYPE tableB JOIN\_CLAUSE

The JOIN\_CLAUSE is where you indicate the relationship between the joined tables.

The 2 main joins are inner & outer (there are subtypes within both).

SELECT forum\_id FROM messages INNER JOIN forums ON messages.forum\_id

= forums.forum\_id

An inner join returns all of the records from the named tables wherever a match is made, e.g. to find every message posted in the MySQL forum, the inner would be written as:

SELECT m.message\_id, m.subject, f.name FROM messages AS m INNER JOIN

Forums AS f ON m.forum\_id = f.forum\_id WHERE f.name = ‘MySQL’;

*The join is selecting 3 columns from the messages table and one column from the forums table under 2 conditions. First, the f.name column must have a value of ‘MySQL’. The second, the forum\_id value in the forums table must match the forum\_id value in the messages table.*

\*More examples of inner joins on p206-207\*

Outer joins

Whereas an inner join returns records based upon making matches between 2 tables, an outer join will return records that are matched by both tables, and will return records that don’t match.

There are 3 outer join subtypes: left, right and full, with left being the most important by far!

SELECT f.\*, m.subject FROM forums AS f LEFT JOIN messages AS m ON

f.forum\_id = m.forum\_id

The most important consideration with left joins is which table gets named first. In this example, all of the forums records will be returned along with all of the messages information, if a match is made.

A right outer join does the opposite of a left outer join: it returns all of the applicable records from the right-hand table, along with matches from the left-hand table.

A full outer join is like a combination of a left outer join and a right outer join. All of the matching records from both tables will be returned, along with all of the records from the left-hand table that do not have matches in the right-hand table, along with all the records from the right-hand table that do not have matches in the left-hand table. \*A full outer join is not often needed\*

\*Examples of outer joins on p209-210\*

\*You can also join 3 or more tables – see p211\*

**Grouping/aggregate functions**

|  |  |
| --- | --- |
| AVG() | The average of the values in a column |
| COUNT() | The number of values in a column |
| GROUP\_CONCAT() | The concatenation of a columns value |
| MAX() | The largest value in a column |
| MIN() | The smallest value in a column |
| SUM() | The sum of all the values in a column |

The AVG(), COUNT() and SUM() functions can also use the DISTINCT keywords so that the aggregation only applies to distinct values i.e. no duplicates.

SELECT COUNT(customer\_id) FROM accounts

* Will return the number of accounts, but…

SELECT COUNT(DISTINCT customer\_id) FROM accounts

* Will return the number of customers that have accounts

The aggregate functions as used on their own return individual values. When aggregate functions are used with a GROUP BY clause, a single aggregate value will be returned for each row in the result set:

SELECT AVG(balance), customer\_id FROM accounts GROUP BY customer\_id

\*Examples on p216-217\*

**Advanced selections**

In order to select data conditionally – similar to an if-else statement:

SELECT IF(condition, return-if-true, return-if-false)

Example:

SELECT IF(gender=’m’, ‘Male’, ‘Female’) FROM people;

Example 2:

INSERT INTO people(gender) VALUES (IF(something=’male’, ‘m’, ‘f’));

Other functions:

|  |  |
| --- | --- |
| GREATEST() | Returns largest value in a list |
| LEAST() | Returns smallest value in a list |
| COALESCE() | Returns first non-null value in a list |
| CASE() | Similar to PHPs switch conditional –  SELECT CASE col1 WHEN value1 THEN return\_this ELSE return\_that END FROM table |

Example of CASE():

SELECT CASE gender WHEN ‘m’ THEN ‘male’ ELSE ‘female’ END FROM people;

The CASE() function can have additional when clauses. The ELSE is also always optional:

SELECT CASE gender WHEN ‘m’ THEN ‘male’ WHEN ‘f’ THEN ‘female’ END FROM people;

You can also write conditions in to a CASE().

\*Examples on p220\*

**Performing FULLTEXT searches**

Altering tables – the ALTER SQL term is primarily used to modify the structure of an existing table i.e. adding, deleting or changing columns, but it also includes the addition of indexes

|  |  |  |
| --- | --- | --- |
| **Desc** | **Query** | **Example** |
| To add a column to a table | ADD COLUMN | ALTER TABLE tablename ADD COLUMN columnname TYPE |
| To change data type & properties of a column | CHANGE COLUMN | ALTER TABLE tablename CHANGE COLUMN columnname TYPE |
| To remove a column from a table | DROP TABLE | ALTER TABLE tablename DROP COLUMN columnname |
| To add an index to a column | ADD INDEX | ALTER TABLE tablename ADD INDEX indexname(columnname) |
| To remove an index | DROP INDEX | ALTER TABLE tablename DROP INDEX indexname |
| To rename a table | RENAME TO | ALTER TABLE tablename RENAME TO new.tablename |

Creating a FULLTEXT index

\*only available on a MyISAM table\*

1. Confirm table type: SHOW TABLE STATUS
2. If table is not of the MyISAM type, change the storage engine: ALTER TABLE tablename ENGINE = MyISAM;
3. Add the FULLTEXT index to the table: ALTER TABLE tablename ADD FULLTEXT(columns);

Performing basic FULLTEXT searches

Once you have established a FULLTEXT index on a column or columns, you can start querying against it, using MATCH…AGAINST in a WHERE conditional:

SELECT \* FROM tablename WHERE MATCH(columns) AGAINST(terms)

To perform FULLTEXT searches

*Using the forums database example*

SELECT subject, body FROM messages WHERE MATCH(body,subject) AGAINST(‘database’);

* This will return records in messages table that have the word ‘database’ in their body or subject.

\*Note that the columns referenced in MATCH must be the same as those in which the FULLTEXT index was made\*

Certain rules apply when FULLTEXT searching:

1. Strings are broken down in to their individual keywords
2. Keywords less than 4 characters long are ignored
3. Very popular words, called stopwords, are ignored
4. If more than 50% of the records match the keywords, no records are returned

\*More examples on p226\*

**Performing Boolean FULLTEXT searches**

The basic FULLTEXT search is nice, but a more sophisticated FULLTEXT search can be accomplished using its Boolean mode. To do so, add the phrase: IN BOOLEAN MODE to the AGAINST clause:

SELECT \* FROM tablename WHERE MATCH(columns) AGAINST(‘+database - MySQL’ IN BOOLEAN MODE);

* In this example, a match will be made if the word ‘database’ is found, and ‘MySQL’ is not present

Boolean mode operators:

|  |  |
| --- | --- |
| + | Must be present in every match |
| - | Must not be present in any match |
| ~ | Lowers a ranking if present |
| \* | Wildcard |
| < | Decrease a words importance |
| > | Increase a words importance |
| “ “ | Must match the exact phrase |
| ( ) | Create sub expressions |

The wildcard character \* matches variations on a word, so cata\* matches catalog, catalina etc..

SELECT \* FROM tablename WHERE MATCH(columns) AGAINST(‘>”web develop” +html ~Javascript’ IN BOOLEAN MODE);

* This would look for records with the phase “web develop” with the word ‘html’ being required and the word ‘javascript’ detracting from a matches relevance.

When using Boolean mode, there are several differences to how FULLTEXT searches work:

1. If a keyword is not precoded by an operator, the word is optional but a match will be ranked higher if it is present
2. Results will be returned even if more than 50% of the records match the search
3. The results are not automatically sorted by relevance

Because of the last fact, you’ll also want to sort the returned records by their relevance as demonstrated in the examples on p228-229

\*see more examples on p228-229\*

**Optimising queries**

Once you have a complete and populated database, and have a sense as to what queries will commonly be run on it, it’s a good idea to take some steps to optimise your queries and your database as a whole. Doing so will ensure you’re getting the best possible performance out of MySQL (and therefore your website).

OPTIMIZE TABLE tablename;

* This command is particularly beneficial after changing a table in an ALTER command, or after a table has had a lot of DELETE queries run on it, leaving virtual gaps among the records.

ANALYZE TABLE tablename

* This command updates the indexes on the table, thereby improving their usage in queries. You could execute it whenever massive amounts of data stored in the table changes (e.g. via UPDATE or INSERT commands)

\*See more examples on p231-233\*

**Performing transactions**

A database transaction is a sequence of queries run during a single session. E.g. you might insert a record into one table, insert another record into another table and maybe run an update. Without using transactions, each individual query takes effect immediately and cannot be undone. With transactions, you can set start and stop points and then enact or retract all of the queries between those points as needed (e.g. if one query failed, all of the queries can be undone).

Example: Transferring £100 from my account to another requires several steps:

1. Confirm I have £100 in my account
2. Decrease £100 from my account
3. Verify the decrease
4. Increase the amount of money in other account by £100
5. Verify the increase worked

If any of the steps failed, all of them should be undone.

\*You must use InnoDB table type in order to perform transactions with MySQL\*

To begin a new transaction:

START TRANSACTION;

* You can now run your queries. Once you have finished, you can either enter COMMIT to enact all of the queries, or ROLLBACK to undo the effect of all queries

You can not perform transactions using PHPMyAdmin, instead run through a PHP script.

\*See examples on p235-236\*

**Database encryption**

If you require data to be stored in an encrypted form that can be decrypted, you’ll want to use AES\_ENCRYPT() & AES\_DECRYPT(). AES\_ENCRYPT() is considered the most secure encryption option.

These functions require 2 arguments:

1. The data being encrypted/decrypted
2. A *salt* argument

*Salt* argument – a string that helps to randomise the encryption (see p237)

INSERT INTO users(username, pass)

VALUES(‘rezaa91’, AES\_ENCRYPT(‘mypass’, ‘nacl19874salt!’));

* The encrypted data returned by the AES\_ENCRYPT() function will be in binary format. To store that data in a table, the column must be defined as one of the binary types (VARBINARY or BLOB)

SELECT \* FROM users WHERE username = ‘rezaa91’ AND

AES\_DECRYPT(pass, ‘nacl19874salt!’) = ‘mypass’;

The exact same salt must be used for both encryption and decryption, which means the salt must be stored somewhere as well. This is safe to store in the same database and even the same table or row.

\*See examples on p238-239\*

**Test**

**Part 1**

1. What is a join?
2. What are the 2 types of joins?
3. What does an inner join do? Give example
4. What does the other type of join do? What are the 3 subtypes of this join?
5. Give an example of an outer join

**Part 2**

1. What are the 6 aggregate functions & what do they do? Give examples
2. What does DISTINCT do?
3. How do you use conditionals in MySQL?
4. What do the following functions do (& give examples):
   * 1. GREATEST()
     2. LEAST()
     3. COALESCE()
     4. CASE()

**Part 3**

1. Give examples of:
   * 1. Adding a column in to an existing table
     2. Changing an existing columns properties and data type
     3. Removing a column from an existing table
     4. Adding an index to an existing column
     5. Removing an index from an existing column
     6. Renaming an existing table
2. How do you confirm what engine type a table is?
3. How do you make a column available to perform FULLTEXT searches?
4. Give basic examples of FULLTEXT searches
5. List Boolean mode operators and give examples
6. Give example of FULLTEXT searches IN BOOLEAN MODE
7. What are the differences between standard FULLTEXT searches and those performed IN BOOLEAN MODE

**Part 4**

1. What does OPTIMIZE TABLE tablename do?
2. What does ANALYZE TABLE tablename do?
3. What are transactions in MySQL? What engine type must you use in order to perform transactions?
4. What functions are used to encrypt and decrypt data?
5. What is a *salt* argument?
6. Give an example of encrypting and decrypting data