**Database for Software Developers, 7. Data Transformation, Grouping and Aggregation – 1**

1. Example scenarios
   1. Concat multiple fields to show them together
   2. Delete carts that were created 3 days ago, but were not created
   3. Find users if the email domain is “myconpany.com”
   4. Show the number of days an application took to get approval
   5. Show a large number with a comma at a specific position
2. The solution is
   1. Build in function
3. What is can they do?
   1. Data formatting
   2. String manipulation
   3. Aggregation
   4. Various conversion
   5. Numeric calculation
   6. Wide range of comparisons
   7. Encryption/decryption
   8. Data and time processing
4. Type of function
   1. String manipulation – CONCAT(), SUBSTRING(), LENGTH(), REPLACE(), UPPER(), LOWER()
   2. Numeric calculations – ROUND(), CEIL(), FLOOR(), ABS(), SQRT()
   3. Data and time – NOW(), CURDATE(), CURTIME(), DATEEDIFF(), DATE\_ADD(), DATE\_FORMAT()
   4. Control flow
      1. IF(), CASE(), COALESCE(), IFNULL()
   5. Cryptography
      1. MD5(), SHA1(), ENCRYPT(), AES\_ENCRYPT(), AES\_DECRYPT()
   6. Aggregation
   7. Search and comparison
   8. Type conversion
   9. Regular expression
   10. JSON functions
   11. Geospatial function
   12. System function
   13. Advanced mathematical
   14. Managing locks
5. String manipulation
   1. CHAR\_LENGTH, LENGTH – the length of a string (in characters/ bytes)
   2. CONCAR, CONCAR\_WS
   3. FORMAT
   4. INSTR – Return the position of the first occurrence of a string in another string. 0 if not match.
   5. LOWER, UPPER
   6. SUBSTR, SUBSTRING, INDEX
   7. TRIM, LTRIM, RTRIM
   8. RIGHT, LEFT
   9. LPAD, RPAD – Right/Left-pads a string with another string, to a certain length
   10. REPLACE
   11. Example
       1. – Get email domain of user input: [example@gmail.com](mailto:example@gmail.com) | output: gamil.com   
          SELECT id, name, SUBSTRING (email, INSTR(email, ‘@’)+1) AS email\_domain   
          FORM users;
       2. Show 30 character in left side  
          SELECT title LEFT (description, 30) AS excerpt, LOWER(REPLACE(title, ‘ ’, ‘-’)) AS url\_slug

FROM products

1. Numeric Calculations
   1. ROUND, FLOOR, CEILEG – round a number with specified strategy
   2. MAX, MIN
   3. SUM
   4. COUNT
   5. RANG
   6. POWER, SQRT
   7. AVG
   8. DIV, MOD – Return the division/reminder 2 number
   9. SIN, COS, TAN, COT
   10. ASIN, ATAN, ATAN2, ACOS
   11. Example
       1. Count number of records, optionally with filters  
          SELECT COUNT(\*)  
          FROM users  
          WHERE role=’admin’
       2. Change 1 out of 3 user to a teacher  
          UPDATE users  
          SET role=’teacher’  
          WHERE MOD(‘id’, 3) = 0;
2. Date and time
   1. Date deference  
      SELECT DATEDIFF(CURDATE(), "2014-12-16");
   2. Get payment with future check date  
      SELECT check\_number, amount  
      FROM payments  
      WHERE payment\_date > CURDATE();
3. Control flow
   1. Display customer type based on credit limit  
      SELECT id, name  
      CASE  
      WHEN credit\_limit > 1000 THEN ‘Premium’  
      WHEN create\_limit > 500 THEN ‘Standard’  
      ELSE ‘New’  
      END AS customer\_type  
      FROM customers;
   2. Get discount eligibility based on current stock   
      SELECT product\_id, IF(strock\_qty > 50, ‘Eligible’, ‘not Eligible’) AS discount FROM products
4. Cryptography / hashing
   1. MD5()
   2. SHA(), SHA1(), SHA2()
   3. AES\_ENCRYPT(), AES\_DECRYPT()
   4. Example
      1. Save user password with MD5 encryption  
         INSERT INTO users(username, password)  
         VALUES(‘goodboy’,MD5(‘the-secret-password’))
      2. Sing document update version with SHA1 encryption  
         UPDATE documents  
         SET conyent\_hash = SHA1(CONCAT(content, username, salt));
      3. Encrypt and decrypt with AES encryption  
         UPDATE customers   
         SET create\_card\_info=AES\_ENCRYPT(‘the-card-number’, ‘encryptionkey’);  
         SELECT customer\_id,AES\_DECRYPT(create\_card\_info, ‘encryptionKey’) AS card\_info FROM customer;

**Database for Software Developers, 7. Data Transformation, Grouping and Aggregation – 2**

1. Aggregation – preforms a function /calculation in a column (across multiple records and returns a single value)
   1. COUNT(), GROUP\_CONCAT(), SUM(), AVG(), MIN(), MAX()
2. Example
   1. SELECT   
      MAX(buy\_price) max\_price,  
      MIN(buy\_price) min\_price,  
      ROUND(AVG(buy\_price),2) average\_price  
      FROM products;
3. Aggregation with filter
   1. SELECT SUM(qty\_in\_stock) FROM PRODUCTS WHERE product\_line=’motorcycles’;
4. Example – 2
   1. SELECT order\_id, SUM (quantity\_ordered \* price\_each) AS total FROM orderd\_items WHERE order\_id = 10010;
5. Example – 3
   1. SELECT COUNT(\*) order\_of\_2003  
      FROM orders  
      WHERE order\_date BETWIN ‘2003-01-01’ AND ‘2003-12-31’;
6. Aggregation and grouping
   1. SELECT SUM(qty\_in\_stick)  
      FROM products;  
      GROUP BY product\_line
7. Usages of DISTINCT – remove the duplicate value
   1. SELECT produst\_line, SUM(qty\_in\_stock), COUNT(vendor)  
      FROM products  
      GROUP BY product\_line;
8. Aggregation with conditional value
   1. Company wants to send gift to all employees. We will use DHL for non-USA employees and USMail for USA employees. How many DHL booking do we need?  
      SELECT  
      SUM(IF(o.country =’USA’, 0, 1)) DHL  
      SUM(IF(o.country =’USA’, 1, 0)) USMail  
      FROM employees e  
      JOIN offices o ON o.code = e.office\_code;
   2. Example-2: shipment code $100 within USA and $300 outside USA how can we get total orders and their shipment costs by sakes person  
      SELECT c.sales\_rep\_id  
      COUNT(DISTINCT o.id) total\_orders,  
      SUM(IF(c.country=’USA’, 100, 300)) total\_shipment\_cost  
      FROM orders o  
      JOIN customers c ON c.id = o.customer\_id  
      GROUP BY c.sales\_rep\_id;
9. Multiple GROUP BY
   1. We have total payment per customer, per year now, we want to see the list of check number with it.
      1. SELECT customer\_id,  
         YEAR(payment\_date) payment\_year,  
         GROUP\_CONCAT(check\_number),  
         SUM(amount) total  
         FROM payments  
         GROUP BY customer\_id, payment\_year;

**Database for Software Developers, 8. Data Indexing and Query Efficiency – 1**

1. Index
   1. A database index is like in a book. It helps MySQL find data faster.
2. Why use index?
   1. Significantly speeds up data retrieval operations
   2. Improve performance of Joining, searching and analytical operations.
   3. Essential for enhancing performance on large databases.
3. Types of index
   1. Primary index
   2. Secondary index
      1. Unique index
      2. Non-unique index
      3. Functional index
4. Primary key
   1. A unique index (always)
   2. NOT NULL (always)
   3. A table will have it
   4. One per table
   5. Data actually stored in this tree
5. Primary key | What should be the data type
   1. CHAR/VARCHAR
   2. UUID/ULID?
   3. INT/BIGINT
   4. INT/BIGINT with UNSIGNED?
6. Primary key | correct way | BIGINT UNSIGNED AUTO\_INCREMENT
   1. Redundancy (storage concern)
   2. Cost of update
   3. Re-balancing B-tree
   4. Obfuscation of ID
7. Planning index
   1. Observe the data-access pattern
   2. Consider all queries being run and their respective access patterns
   3. Consider the entire query – includes sorting, grouping, and joining.
8. How to create index in products table
   1. ALTER TABLE products.ADD\_INDEX(name);
9. How to check is the table including index
   1. SHOW INDEXS FROM products;
10. Planning indexs
    1. New index to support query/ change query to make use of index
    2. Minimum indexs / too much indexs
11. Drop index
    1. DROP INDEX name on products

**Database for Software Developers, 8. Data Indexing and Query Efficiency - 2**

1. Functional index
   1. ALTER TABLE users ADD INDEX joining\_year((YEAR(created\_at)));
   2. ALTER TABLE users ADD INDEX email\_domain((SUBSTRING(email, INSTR(email, ‘@’)+1)));
2. Composite index
   1. SELECT name, price, stock\_quantity, shipment\_type  
      FROM products  
      WHERE shipment\_type = ‘physical’  
      AND ‘name’ LIKE ‘Ultra%’  
      AND stock\_quantity > 0
   2. ALTER TABLE products ADD INDEX search\_q (shipment\_type, name, stock\_quantity);
3. Composite index | General rule of using composite index
   1. ALTER TABLE products ADD INDEX multicol\_idx (col\_a, col\_b, col\_c);  
      WHERE col\_a = ‘X’ AND col\_b = 100 AND col\_c = ‘Y’;  
      WHERE col\_a = ‘X’;  
      WHERE col\_a = ‘X’ AND col\_b = 100;  
      WHERE col\_a = ‘X’ AND col\_c = ‘Y’  
      WHERE col\_a = ‘X’ AND col\_b > 100 AND col\_c = ‘Y’
4. FULLTEXT index
   1. Let’s find films with the words victory and ‘Drama’ and …
   2. Too much condition
      1. SELECT \* FROM film\_text WHERE title LIKE ‘%victory%’   
         OR description LIKE ‘%Vicotry%’   
         OR title LIKE ‘%Drama%’   
         OR description LIKE ‘%Drama%’   
         OR …;
   3. Optimize query
      1. Create INDEX
         1. ALTER TABLE filem\_text ADD FULLTEXT INDEX search\_film (title, description);
      2. Match query
         1. SELECT \*  
            FORM film\_text  
            WHERE MATCH (title, description) AGAINST(‘Any number of keywords’);
      3. Supports InnoDB or MyISAM storage engine
      4. Can be user with only CHAR, VARCHAR and TEXT columns
      5. Words with ¾ characters and stop-words are ignored (by default)
5. Query optimization | and identifying slow Queries
   1. Identifying slow queries
      1. Considering the expectation and business requirements!
      2. Consider the scale of the application
      3. Consider the use case of customer
   2. Are you using an ORM
      1. Check if they’re right write queries are being executed
      2. Beware of n+1 Query issues
   3. Enable show query log
      1. Turn slow query logging ON

SET GLOBAL slow\_query\_log = 'ON';

* + 1. Keeping Slow query log in file

SET GLOBAL slow\_query\_log\_file = '/tmp/slow\_queries.log';

* + 1. Keeping Slow query log in table

SET GLOBAL log\_output = 'table';

* + 1. Additional settings

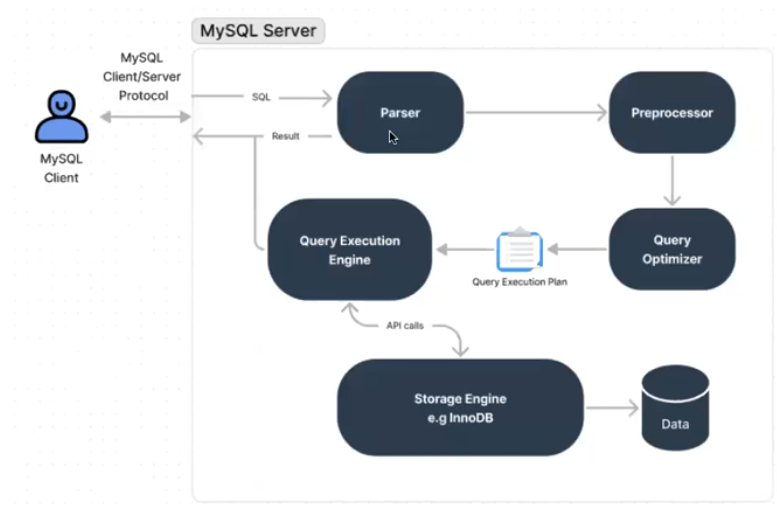
SET GLOBAL log\_queries\_not\_using\_indexes = 'ON';

SET GLOBAL long\_query\_time = 5;

* + 1. Turn slow query logging off  
       SET GLOBAL slow\_query\_log = 'OFF';
  1. Query profiling
     1. SET SESSION profiling = 1;
     2. Execute you queries  
        SELECT \* FROM film;  
        SELECT \* FROM customer;
  2. Make use of EXPLAIN  
     EXPLAIN   
     SELECT first\_name, last\_name, city, country

FROM customer   
INNER JOIN address USING (address\_id)

INNER JOIN city USING (city\_id)   
INNER JOIN country USING (country\_id);

* 1. Optimizing query   
     

1. Fetch as fewer data as possible
   1. SELECT \* FROM customer;
   2. SELECT id, first\_name, last\_name FROM customer;
2. Optimize JOIN query
   1. Use INNER JOIN and LEFT JOIN
   2. Ensure the joining columns are indexed
   3. Use EXPLAIN to verify the right indexs are being used
   4. Choose the order of joining wisely – try to start with fewer rows
3. Group BY and ORDER BY
   1. Try to use same table for grouping and ordering
   2. Ordering and grouping can use index
   3. Try to use existing index by changing order column (if possible)
4. Make use of indexes
   1. WHERE conditions should use index where possible
   2. GROUP BY columns should use index where possible
   3. ORDER BY columns should use index where possible
   4. Try using composite index for multiple conditions
5. Make use of caching
   1. Use Redis, Memcached or even filesystem
   2. Beware of serving stale information from cache
   3. Invalidate cache keys wisely
6. Make use of pre-calculated results
   1. Use materialized views for statistical data
   2. Use flat report tables with pre-populated data
   3. User supporting systems – redis, mongoDB

**Database for Software Developers, 9. Subqueries, Views, and Search - 2**

1. Subquery