Cheatography

Fundamentals of Data Management Cheat Sheet

by NanoSeeker via cheatography.com/31128/cs/9400/

Regex - Quantifiers

*	0 or more	{3}	Exactly 3
+	1 or more	{3,}	3 or more
?	0 or 1	{3,5}	3, 4 or 5

Regex - Character Classes

\c	Control character
\s	White space
\S	Not white space
\d	Digit
\D	Not digit
\w	Word
\W	Not word
\x	Hexadecimal digit
\O	Octal digit

Regex - Groups and Ranges

.	Any character except new line (\n)
(a|b)	a or b
(...)	Group
(?:...)	Passive (non-capturing) group
[abc]	Range (a or b or c)
[^abc]	Not (a or b or c)
[a-q]	Lower case letter from a to q
[A-Q]	Upper case letter from A to Q
[0-7]	Digit from 0 to 7
\x	Group/subpattern number "x"

Ranges are inclusive.

Regex - Examples

Username

```
/^[a-z0-9_-]{3,16}$/
```

Hex Value

```
/^#?([a-f0-9]{6}|[a-f0-9]{3})$/
```

Email

```
/^([a-z0-9_\.-]+)@([\da-z\.-]+)\.([a-z\.]{2,6})$/
```

Regex - Examples (cont)

URL

```
/^(https?:\/\/)?([\da-z\.-]+)\.([a-z\.]{2,6})([\/\w \.-]*)\/?$/
```

IP Address

```
/^(?:(?:25[0-5]|2[0-4][0-9]|[01]?[0-9][0-9]?)\.){3}(?:25[0-5]|2[0-4][0-9]|[01]?[0-9][0-9]?)$/
```

12 Hour Time

```
/^(1[0-2]|0?[1-9]):([0-5][0-9])\s?([ap][m]|[AP][M])
```

SQL - Data Types

CHAR	String (0 - 255)
VARCHAR	String (0 - 255)
TINYTEXT	String (0 - 255)
TEXT	String (0 - 65535)
MEDIUMTEXT	String (0 - 16777215)
LONGTEXT	String (0 - 4294967295)
TINYINT x	Precision 3
SMALLINT x	Precision 5
MEDIUMINT x	Precision 7
INTEGER x	Precision 10
BIGINT x	Precision 19
FLOAT	Decimal (precise to 23 digits)
DOUBLE	Decimal (24 to 53 digits)
DECIMAL	"DOUBLE" stored as string
DATE	YYYY-MM-DD
DATETIME	YYYY-MM-DD HH:MM:SS
TIMESTAMP	YYYYMMDDHHMMSS
TIME	HH:MM:SS

Integers (marked x) that are "UNSIGNED" have the same range of values but start from 0 (i.e., an UNSIGNED TINYINT can have any value from 0 to 255).

SQL - String Functions

Compare strings	STRCMP("str1","str2")
Convert to lower case	LOWER("str")
Convert to upper case	UPPER("str")
Left trim	LTRIM("str")
Substring of a string	SUBSTRING("str","inx1","inx2")
Concatenate	CONCAT("str1","str2")

SQL - Calculation Functions

Count rows	COUNT(col)
Average	AVG(col)
Minimum value	MIN(col)
Maximum value	MAX(col)
Sum of values	SUM(col)

SQL - SELECT Queries

select all columns

```
SELECT * FROM tbl;
```

select some columns

```
SELECT col1, col2 FROM tbl;
```

select only unique records

```
SELECT DISTINCT FROM tbl WHERE condition;
```

column alias with AS

```
SELECT col FROM tbl AS newname;
```

order results

```
SELECT * FROM tbl ORDER BY col [ASC | DESC];
```

group results

```
SELECT col1, SUM(col2) FROM tbl GROUP BY col1;
```

SQL - Creating and Modifying

create a database

```
CREATE DATABASE db_name;
```

select a database

```
USE db_name;
```

create a new table

```
CREATE TABLE tbl (field1, field2);
```

insert data into a table

```
INSERT INTO tbl VALUES ("val1", "val2");
```

delete a row

```
DELETE * FROM tbl WHERE condition;
```

add a column from a table

```
ALTER TABLE tbl ADD COLUMN col;
```

remove a column from a table

```
ALTER TABLE tbl DROP COLUMN col;
```

make a column a primary key

```
ALTER TABLE tbl ADD PRIMARY KEY (col);
```

return only 1 row matching query

```
... LIMIT = 1
```

amend the values of a column

```
UPDATE table SET column1="val1" WHERE ...
```

clear all the values, leaving the table structure

```
TRUNCATE TABLE tbl;
```

delete the table

```
DROP TABLE tbl;
```

delete the database

```
DROP DATABASE db_name;
```

SQL - Keys

Primary Key

Must contain UNIQUE values
Column cannot contain NULL values

Composite Key

Primary key with multiple fields

Foreign Key

A FOREIGN KEY in one table points to a PRIMARY KEY in another table

SQL - Normal Forms

First

> There are no repeating groups
> All data values are atomic
> Each field has a unique name
> It has a primary key

Second

> No partial dependencies
> All non key attributes are dependent on all parts of the primary key

Third

> All non key attributes are not dependent on any other non-key attributes

SQL - Create table with auto-increment primary key

```
CREATE TABLE table_name (
id INT AUTO_INCREMENT,
column VARCHAR(2),
column VARCHAR(32),
PRIMARY KEY (id)
);
```

SQL - Tables

```
CREATE TABLE IF NOT EXISTS Student
(
stud_id INTEGER unsigned not null,
stud_name VARCHAR(30),
stud_phone INTEGER unsigned,
stud_date_of_birth DATE,
stud_city VARCHAR(30),
stud_address VARCHAR(30),
stud_postcode SMALLINT unsigned,
PRIMARY KEY(stud_id)
);
CREATE TABLE IF NOT EXISTS Subj_Enrolment
(
stud_id INTEGER unsigned not null,
subj_code VARCHAR(8) not null,
```

SQL - Tables (cont)

```
semester SMALLINT unsigned not null,
year SMALLINT unsigned not null,
comment VARCHAR(100),
PRIMARY KEY(stud_id, subj_code, semester, year),
FOREIGN KEY (stud_id) REFERENCES Student (stud_id),
FOREIGN KEY (subj_code) REFERENCES Subject
(subj_code)
);
```

SQL - Examples

Modify the STUDENT table, add new column called gender and add check constraints to make sure that the values stored are either Male or Female

> ALTER TABLE Student ADD Gender CHAR(6) CHECK (Gender in ('Male','Female')) (One line)

Modify the SUBJ_ENROLMENT table, delete the comment column

> ALTER TABLE Subj_Enrolment DROP COLUMN comment (One line)

The phone number should be concatenated using area code and phone number in the format +123 555-2686, where 123 is the area code and 555-2686 is the phone number. customers who live in Washington state (WA) or Oregon

> SELECT CustFirstName, CustLastName,
> CONCAT('+',CustAreaCode,' ', CustPhoneNumber) AS NationalNumber (One line)
> FROM Customers
> WHERE CustState = 'WA' OR CustState = 'OR'; (One line)

Write a query that shows the number of customers for per state

> SELECT COUNT(*) as Customers, CustState
> FROM Customers
> GROUP BY CustState;

Insert data to table

> Insert into PurchasedItem (purchaseID, itemNo, productName, orderedqty, quotedPrice) VALUES (last_insert_id(), 1, 'Cricket bat', 2, 80.50); (One line)

CLUB NAMES which have been founded before 2010

> SELECT CName, Founded
> FROM CLUB
> WHERE Founded < 2010;

SQL - Examples (cont)

Number of each ClubID, High to low sorted by count

```
SELECT COUNT() as 'COUNT()', ClubID
FROM STUDENT
GROUP BY ClubID
ORDER BY COUNT(*) DESC;
```

Join Example

```
SELECT S.STUDENTID, S.SNAME, C.CNAME
FROM STUDENT S
NATURAL JOIN CLUB C
WHERE S.GENDER = 'MALE' OR S.AGE <24
ORDER BY S.STUDENTID DESC
```

SQL - Joins

INNER JOIN	returns only where match in both tables
OUTER JOIN	also returns non-matching records from both tables
LEFT JOIN	also returns non-matching records from left table
RIGHT JOIN	also returns non-matching records in right table

XML - Example

```
<james_reading_list>
  <book>
    <title>Fifty shades of grey</title>
    <author>E.L.James</author>
    <status>
      <read>yes</read>
      <time>May 2016</time>
      <outcome>Did not like it very much</outcome>
    </status>
  </book>
  <book>
    <title>The grass is singing</title>
    <author>Doris Lessing</author>
    <status>
      <read>yes</read>
      <time>June 2016</time>
      <outcome>Enjoyed it quite a bit</outcome>
    </status>
```

XML - Example (cont)

```
  </book>
  <book>
    <title>A short history of nearly
everything</title>
    <author>Bill Bryson's</author>
    <status>
          <read>yes</read>
          <time>July 2016</time>
          <outcome>Found it very
informative</outcome>
    </status>
  </book>
  <book>
    <title>JSON in 24 hours</title>
    <author>Peter Settler</author>
    <status>
      <read>no</read>
      <time>Later in the year</time>
      <outcome>N/A</outcome>
    </status>
  </book>
  <book>
    <title>Miss Smilla's feeling for snow</title>
    <author>Peter Hoeg's</author>
    <status>
      <read>no</read>
      <time>TBD</time>
      <outcome>N/A</outcome>
    </status>
  </book>
</james_reading_list>
```

XQuery - Examples use XML

XQuery that returns all the book titles

```
for $b in /james_reading_list/book/title
return $b
```

XQuery FLWOR expression that lists only the titles and authors of the books that James hasn't read yet

```
for $b in /james_reading_list/book
where $b/status/read = "no"
return $b/(title, author)
```

JSON - Objects

```
var myObject = {
  "first": "John",
  "last": "Doe",
  "age": 39,
  "sex": "male",
  "salary": 70000,
  "registered": true
};
```

JSON - Access object properties

myObject.sex	returns "male"
myObject["age"]	returns 39
myObject[0]	returns "John"
myObject.something	returns undefined
myObject[6]	returns undefined

JSON - Array of objects

```
var myArray = [
  {
    "first": "John",
    "last": "Doe",
    "age": 39,
    "sex": "male",
    "salary": 70000,
    "registered": true
  },
  {
```

JSON - Array of objects (cont)

```
    "first": "Jane",
    "last": "Smith",
    "age": 42,
    "sex": "female",
    "salary": 80000,
    "registered": true
  },
  {
    "first": "Amy",
    "last": "Burnquist",
    "age": 29,
    "sex": "female",
    "salary": 60000,
    "registered": false
  }
];
```

JSON - Access Object Array Elements

myArray[0]	returns { "fist": "John", "last": "Doe" ... }
myArray[1]	returns { "fist": "Jane", "last": "Smith" ... }
myArray[1].first	returns "Jane"
myArray[1][2]	returns 42
myArray[2].registered	returns false
myArray[3]	returns undefined
myArray[3].sex	error: "cannot read property..."

JSON - Arrays

```
var myArray = [
  "John",
  "Doe",
  39,
  "M",
  70000,
  true
];
```

JSON - Nested objects and arrays

```
var myObject = {
  "ref": {
    "first": 0,
    "last": 1,
    "age": 2,
    "sex": 3,
    "salary": 4,
    "registered": 5
  },
  "jdoe1": [
    "John",
    "Doe",
    39,
    "male",
    70000,
    true
  ],
  "jsmith1": [
    "Jane",
    "Smith",
    42,
    "female",
    80000,
    true
```

JSON - Nested objects and arrays (cont)

```
  ]
};
```

JSON - Access array elements

myArray[1]	returns "Doe"
myArray[5]	returns true
myArray[6]	returns undefined

JSON - Access Nested Array Elements

myObject.ref.first	returns 0
myObject.jdoe1	returns ["John", "Doe", 39 ...]
myObject[2]	returns ["Jane", "Smith", 42 ...]
myObject.jsmith1[3]	returns "female"
myObject[1][5]	returns true
myObject.jdoe1[myObject.ref.last]	returns "Doe"
myObject.jsmith1[myObject.ref.age]	returns 42