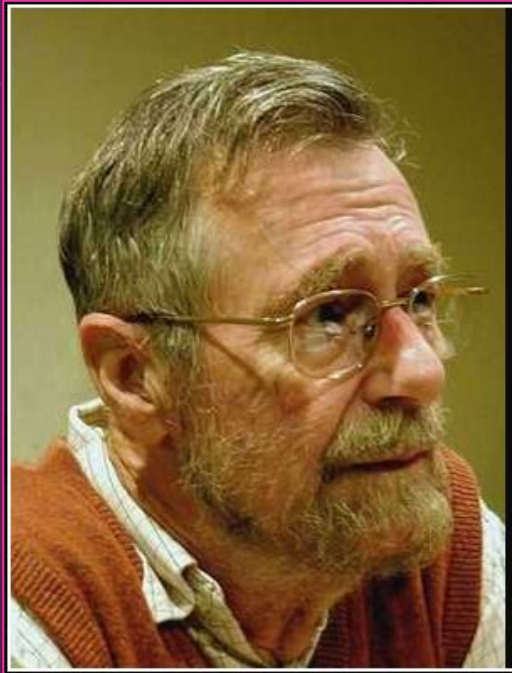
GlideRecord

Interacting with the ServiceNow Database

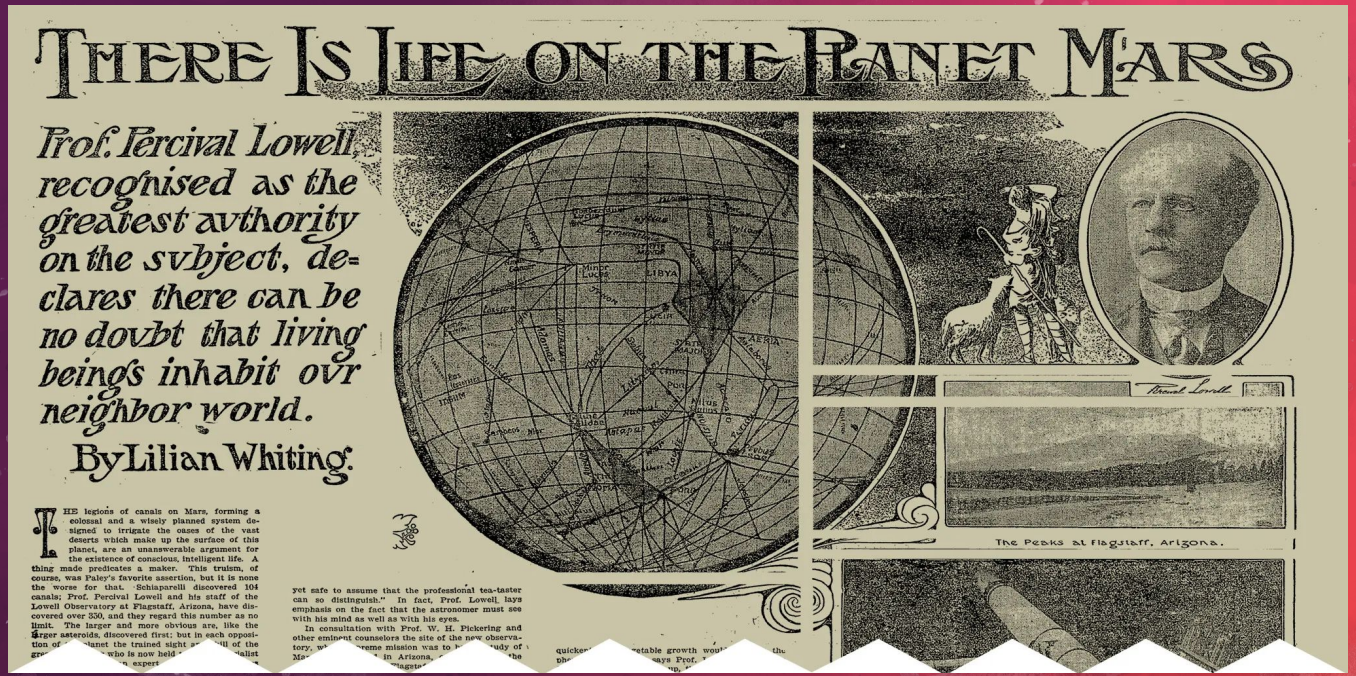
The Paradox of Programming



If debugging is the process of removing software bugs, then programming must be the process of putting them in.

— Edsger Dijkstra —

I don't see why the hell not



THERE IS LIFE ON THE PLANET MARS

Prof. Percival Lowell, recognised as the greatest authority on the subject, declares there can be no doubt that living beings inhabit our neighbor world.

By Lilian Whiting.

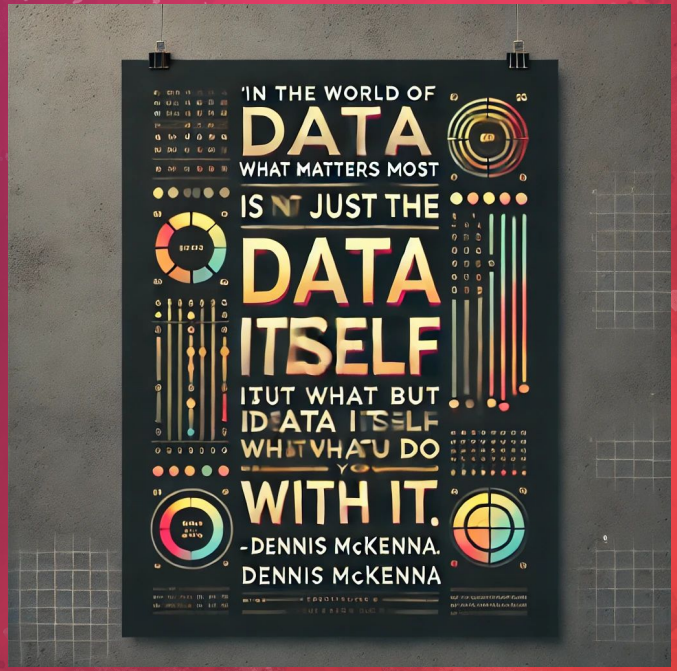
THE legions of canals on Mars, forming a colossal and a wisely planned system designed to irrigate the oases of the vast deserts which make up the surface of this planet, are an unanswerable argument for the existence of conscious, intelligent life. A thing made predicates a maker. This truism, of course, was Paley's favorite assertion, but it is none the worse for that. Schiaparelli discovered 104 canals; Prof. Percival Lowell and his staff of the Lowell Observatory at Flagstaff, Arizona, have discovered over 330, and they regard this number as no limit. The larger and more obvious are, like the larger asteroids, discovered first; but in each opposition of [illegible] planet the trained sight [illegible] of the [illegible] who is now held [illegible] expert [illegible] yet safe to assume that the professional tea-taster can so distinguish." In fact, Prof. Lowell lays emphasis on the fact that the astronomer must see with his mind as well as with his eyes.

In consultation with Prof. W. H. Pickering and other eminent counselors the site of the new observatory, [illegible] supreme mission was to [illegible] study of [illegible] Mars, [illegible] in Arizona, [illegible] the [illegible] quickest [illegible] etable growth would [illegible] the [illegible] phenomena [illegible] says Prof. [illegible]

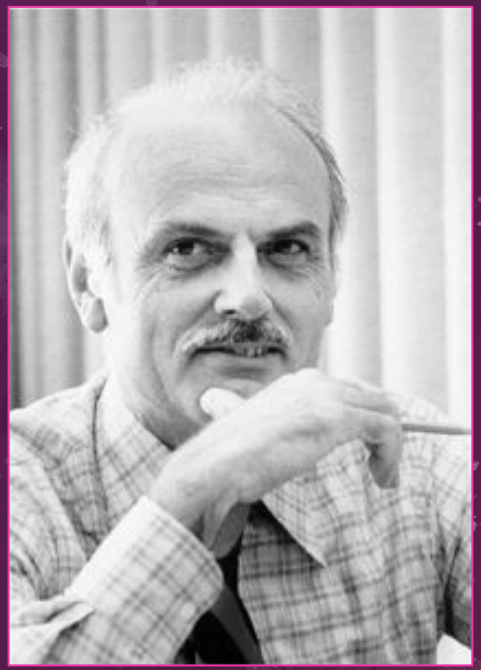
Moving Beyond Raw Information

"It is a capital mistake to theorize before one has data."

Sherlock Holmes



More Than Just Storing Data



"The relational model is not just a way to store data but a way to think about data."

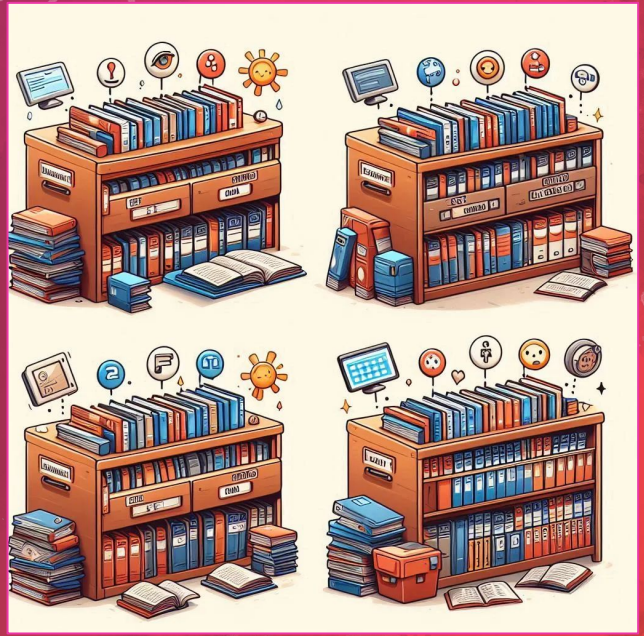
- **Edgar Frank "Ted" Codd** was an English computer scientist who, while working for IBM, invented the relational model for database management, the theoretical basis for relational databases and relational database management systems.

Where Data Lives

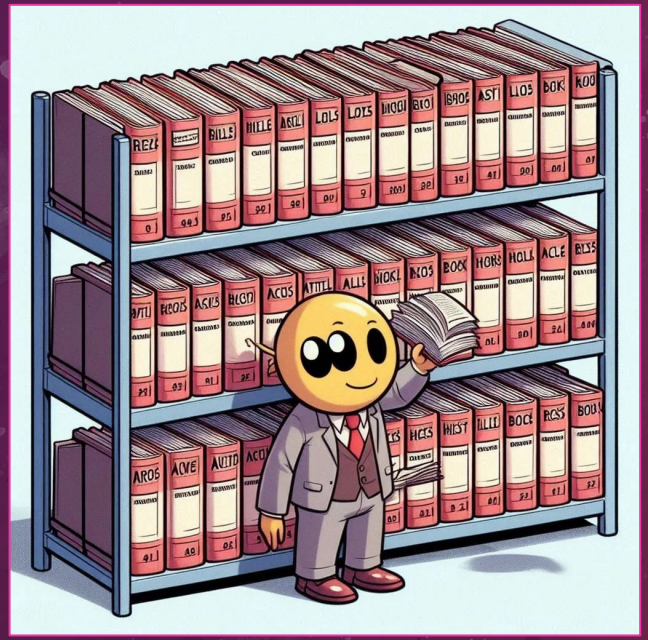
- A database is an organized collection of data that is stored, managed, and accessed electronically.
- Databases are designed to efficiently handle large amounts of information and allow users to retrieve, update, or manipulate data in a structured manner.
- The entire database is like the library.
- Just as a library houses a collection of books, a database contains tables

Your Favorite Book Collection

- A table in a database is a collection of related data entries organized into rows and columns.
- Each row represents a single record or entity, while each column represents a specific attribute or field related to that entity.
- Tables in a database are like bookshelves in the library.
- Each table stores specific information, much like each bookshelf holds books of a particular genre or category.
- Different sections, such as Fiction, Non-Fiction, Reference, etc., correspond to different tables within the database.



Where Every Row is a Story!



- Rows on a table are like books on a shelf.
- Each row represents a single record or entry, similar to how each book represents a specific item within the category.
- Columns in a table are like the specific details of each book.
- Each column represents a particular attribute or information about the records, similar to how each piece of book information (title, author, etc.) describes the book.

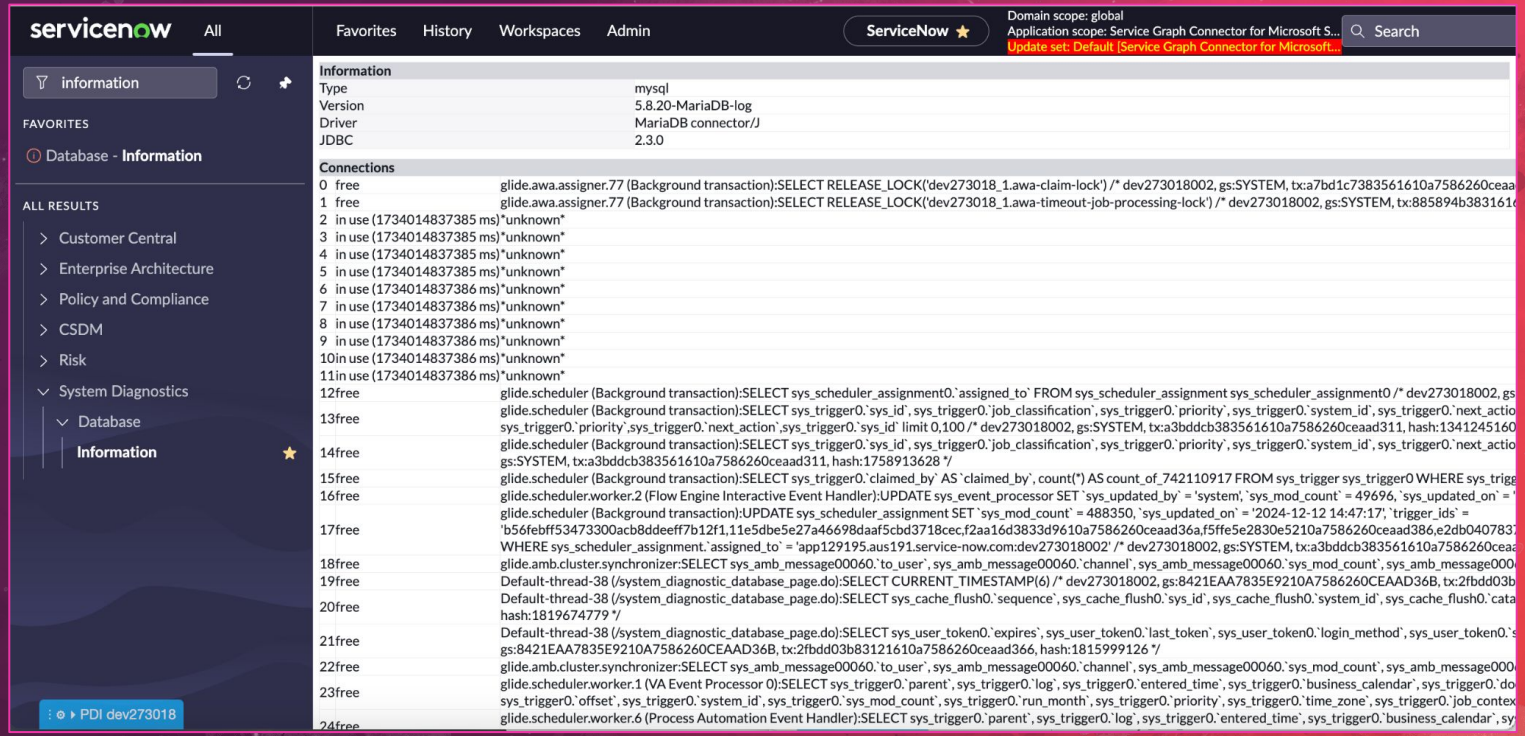
The magical language of databases

- SQL was one of the first commercial languages to use Edgar F. Codd's relational model.
- SQL (Structured Query Language) is a programming language specifically designed for managing and manipulating relational databases.
- It provides a standard way to perform tasks such as querying, updating, inserting, and deleting data within a database.

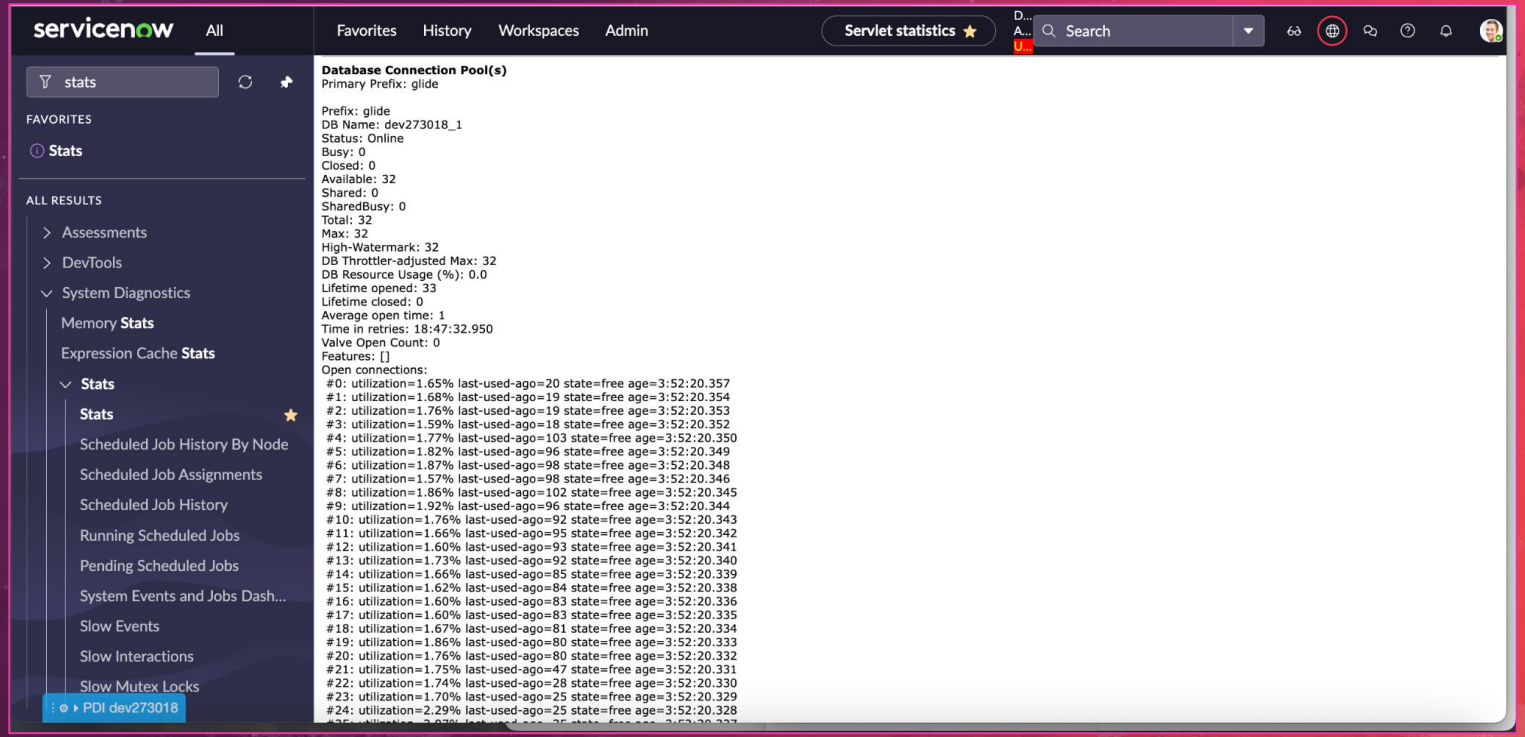
Where SQL Gets a Day Off!

- ServiceNow instances are hosted on infrastructure that uses MariaDB databases to store all platform data.
- The database holds tables, records, configurations, and platform metadata.
- Users interact with a high-level interface (e.g., forms, lists, dashboards) or APIs (like GlideRecord) that abstract database complexity.
- ServiceNow shields users from writing SQL queries, providing a low-code/no-code environment for data handling.

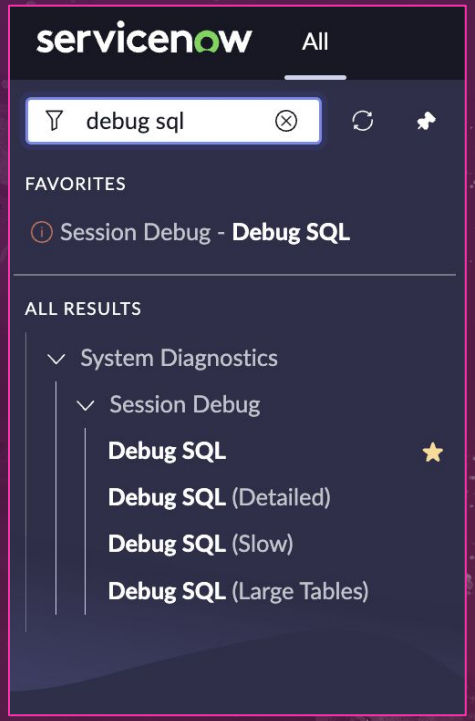
What's Your Instance Running?

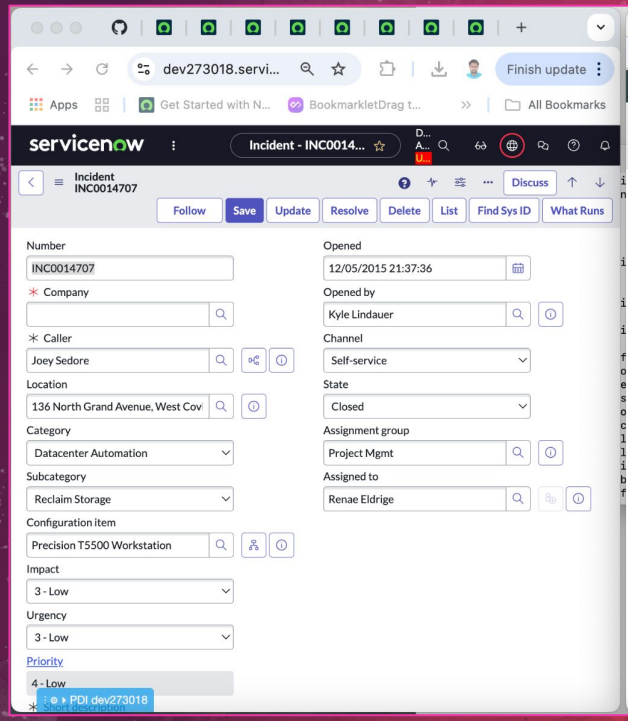


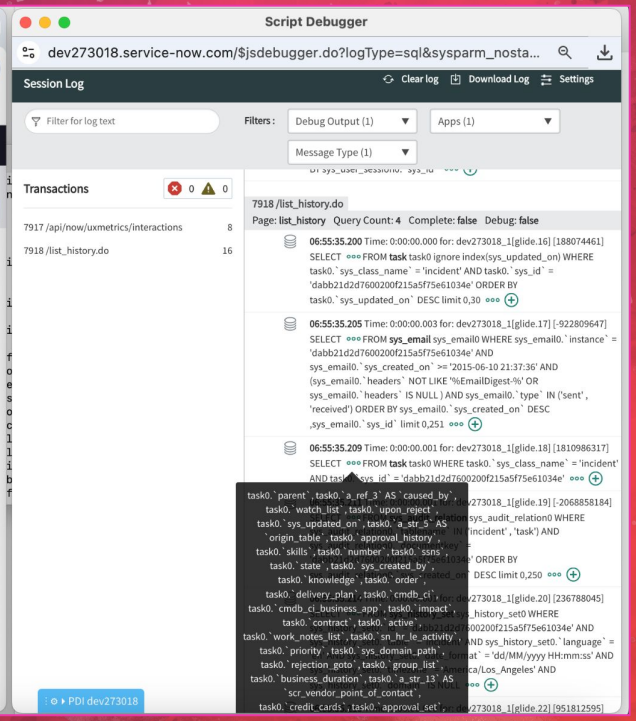
What's Your Instance Running?



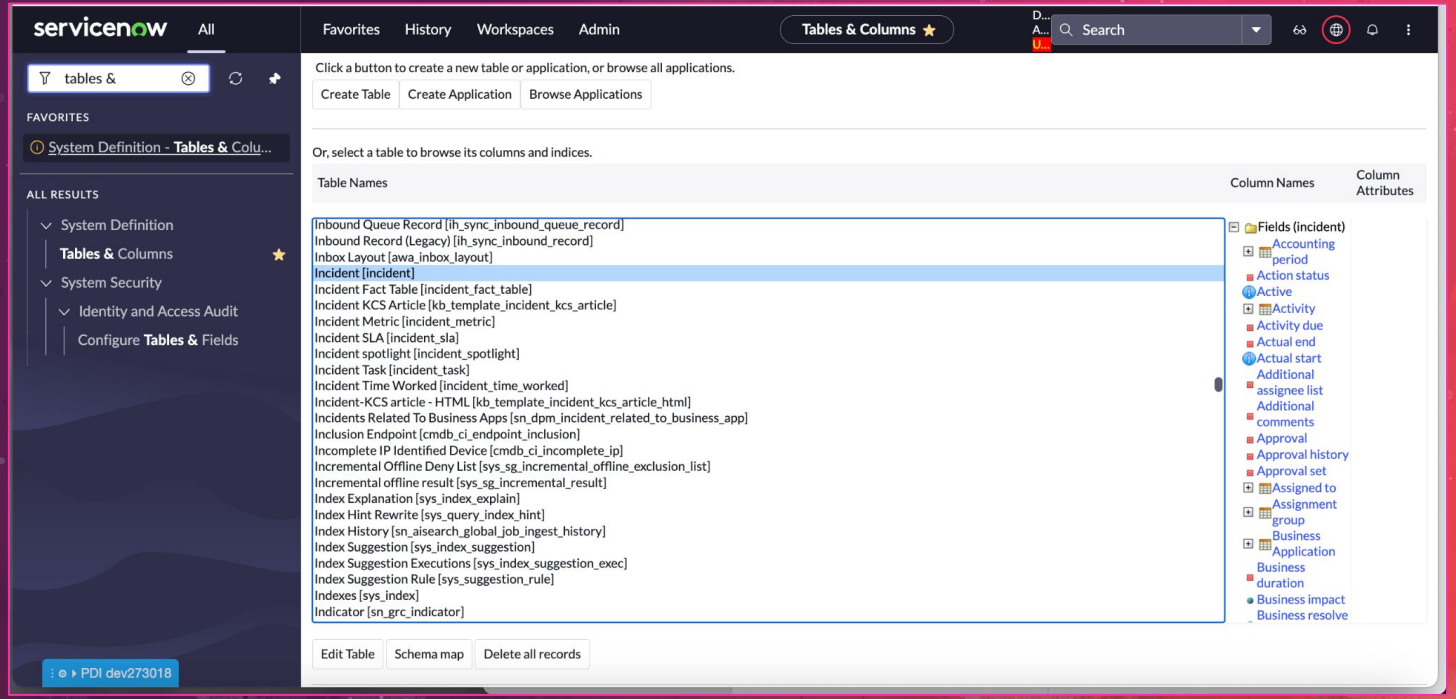
What's Your Instance Running?



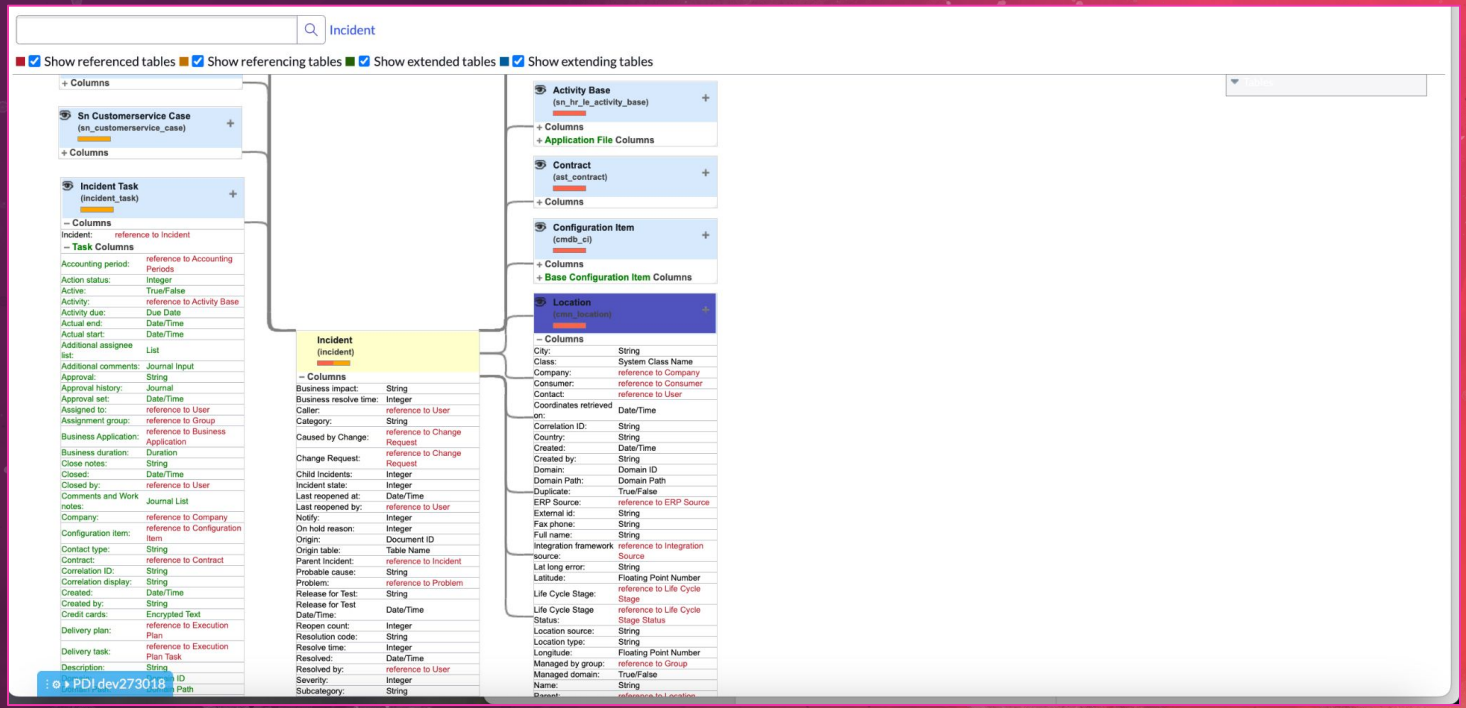




What's Your Instance Running?



What's Your Instance Running?



The Story of a Sea Lion

- MariaDB is a community-developed, commercially supported fork of the MySQL relational database management system (RDBMS), intended to remain free and open-source software under the GNU General Public License.
- Development is led by some of the original developers of MySQL, who forked it due to concerns over its acquisition by Oracle Corporation in 2009.
- MariaDB is intended to maintain high compatibility with MySQL, with exact matching with MySQL APIs and commands, allowing it in many cases to function as a drop-in replacement for MySQL.

The Story of a Sea Lion

- **Ulf Michael Widenius** (born 3 March 1962), also known as **Monty**, is a Finnish software programmer.
- He is the main author of the original version of the open source MySQL database, a founding member of the MySQL AB company and CTO of the MariaDB Corporation AB.

""It happened when Monty and his older daughter My were snorkeling on one of the islands in the Galapagos. Something big, brown and fast suddenly appeared at an arm's distance, laughing in their faces. Fond memories of this fast and funny creature, scaring the tourists, popped into Monty's mind when asked picking a logo for MariaDB. He wanted to adhere to the tradition of animals as symbols of Open Source projects."

- Widenius has three children - **My, Max,** and **Maria** - who inspired the names for MySQL,MaxDB and the MySQL-Max distribution, and MariaDB.

SELECT Query: Finding Books

SELECT title, author FROM Books WHERE genre = 'Science Fiction';

- You want to find all science fiction books and learn their titles and authors.
- SELECT: Tell the librarian what information you want (titles and authors).
- FROM: Specifies the section you're looking in (Books shelf).
- WHERE: A filter is added to find only Science Fiction books.

GlideRecord

- The GlideRecord class is the way to interact with the ServiceNow database from a script.
- GlideRecord interactions start with a database query.
- The generalized strategy is:
  a. Create a GlideRecord object for the table of interest.
  b. Build the query condition(s).
  c. Execute the query.
  d. Apply script logic to the records returned in the GlideRecord object.

GlideRecord(String tableName)

- Creates an instance of the GlideRecord class for the specified table

```
var incGr = new GlideRecord('incident');
```

query()

- **Runs a query against the table based on the filters** specified by query methods such as addQuery() and addEncodedQuery().
- This method fails if there is a field in the table called "query".
- If that is the case, use the method _query().
- To run queries in a domain-separated instance, use the method queryNoDomain().

```
var incGr = new GlideRecord('incident');
incGr.query();
```

next()

- Moves to the next record in the GlideRecord.
- Use this method to iterate through the records returned by a GlideRecord query.
- This method fails if there is a field in the table called "next". If that is the case, use the method _next().
- There are several strategies for iterating through returned records.
- The next() method and a **while** loop iterates through all returned records to process script logic:

```
var incGr = new GlideRecord('incident');
incGr.query();
while (incGr.next()) {
}
```

next()

- The next() method and an **if** processes only the first record returned.

```
var incGr = new GlideRecord('incident');
incGr.query();
if (incGr.next()) {
}
```

LIMIT: Getting a Subset of Books

SELECT title, author FROM Books ORDER BY year DESC LIMIT 5;

- You want to list the 5 most recently published books.
- SELECT title, author: Tells the librarian you want titles and authors. FROM Books: Specifies the section.
- ORDER BY year DESC: Orders the books by publication year in descending order (newest first).
- LIMIT 5: Only shows the top 5 results.

setLimit(Number limit)

- Sets the maximum number of records to return in the GlideRecord from a query.

```
var incGr = new GlideRecord('incident');
incGr.setLimit(3);
incGr.query();
while (incGr.next()) {


}
```

addExtraField(String dotWalkedField)

- Queries one or more dot-walked fields from a form or script in a single request.
- The `addExtraField()` method allows you to query dot-walked fields in a single database request, rather than perform multiple queries per dot-walked element in a form or script (which requires multiple round trips to the database).
- Levels of field names are separated by dots (periods).
- The format of dotWalkedField follows the left-to-right order of fields in a dotwalked form or script.
- The `addExtraField()` method optimizes the querying of the dot-walked fields.

addExtraField(String dotWalkedField)

```
var incGr = new GlideRecord('incident');
incGr.addEncodedQuery("active=true");
incGr.addExtraField('caller_id.location.name')
incGr.query();
if (incGr.next()) {
   gs.info(incGr.caller_id.location.name);
}
```

Retrieve values from records

- In most cases, don't use dot-walking to get values from a record.
- Dot-walking retrieves the entire object instead of the field value.
- Retrieving the object uses more storage and might cause undesirable results when used in arrays or in Service Portal.
- Instead of retrieving the entire object, you can use one of the following methods to copy the field values:
  - getValue()
  - getDisplayValue()

Retrieve values from records

- If dot-walking through a GlideElement object is necessary, use the toString() method to retrieve values.
- For example, you might need the current caller's manager sys_id to set another reference field.

```
var incGr = new GlideRecord('incident');
incGr.query();
if (incGr.next()) {
 var caller=incGr.caller_id.toString();
   gs.info(typeof caller);
}
```

getValue(String fieldName)

- Retrieves the string value of a specified field or the string value of an attribute in a dynamic attribute store.
- Returns null if the field is empty or the field doesn't exist.
- Boolean values return as "0" and "1" string values instead of false and true.

```
var incGr = new GlideRecord('incident');
incGr.setLimit(3);
incGr.query();
while (incGr.next()) {
   gs.info(incGr.getValue("priority"));
}
```

getDisplayValue(String name)

- Retrieves the display value for the current record or the display value of an attribute in a dynamic attribute store.

```
var incGr = new GlideRecord('incident');
incGr.setLimit(3);
incGr.query();
while (incGr.next()) {
   gs.info(incGr.getDisplayValue("priority"));
}
```

getUniqueValue()

- Gets the primary key of the record, which is usually the sys_id unless otherwise specified.

```
var incGr = new GlideRecord('incident');
incGr.setLimit(3);
incGr.query();
while (incGr.next()) {
   gs.info(incGr.getUniqueValue());
}
```

getRecordClassName()

- Retrieves the class (table) name for the current record.

```
var incGr = new GlideRecord('incident');
incGr.query();
if (incGr.next()) {
   gs.info(incGr.getRecordClassName());
}
```

addQuery(String name, Object operator, Object value)

- Provides the ability to build a request, which when executed, returns the rows from the specified table that match the request.
- If you're familiar with SQL, this method is similar to the "where" clause.
- You can make one or more addQuery() calls in a single query.
- For this method the queries are AND'ed.
- addQuery() is typically called with three parameters; table field, operator, and comparison value.
- It can be called with only two parameters, table field and comparison value, The operator in this case is assumed to be "equal to".

addQuery(String name, Object operator, Object value)

```
var incGr = new GlideRecord('incident');
incGr.addQuery("active","=",true);
incGr.query();
if (incGr.next()) {
gs.info(incGr.getDisplayValue('number'));
}
```

```
var incGr = new GlideRecord('incident');
incGr.addQuery("active","=",true);
incGr.query();
if (incGr.next()) {
gs.info(incGr.getDisplayValue('number'));
}
```

addQuery(String name, Object operator, Object value)

- The addQuery operators are:
  - Numbers: =, !=, >, >=, <, <=
  - Strings: =, !=, STARTSWITH, ENDSWITH, CONTAINS, DOES NOT CONTAIN, IN, NOT IN, INSTANCEOF
- When there are multiple queries, each additional clause is treated as an AND.
- Queries with no query conditions return all records from a table.
- If a malformed query executes in runtime, all records from the table are returned.
- An incorrectly constructed encoded query, such as including an invalid field name, produces an invalid query.
- When the invalid query is run, the invalid part of the query condition is dropped, and the results are based on the valid part of the query, which may return all records from the table.
- You can set the `glide.invalid_query.returns_no_rows` system property to `true` to have queries with invalid encoded queries return no records.

addActiveQuery()

- Adds a filter to return active records.

```
var incGr = new GlideRecord('incident');
incGr.addActiveQuery();
incGr.query();
if (incGr.next()) {
   gs.info(incGr.getDisplayValue('number'));
}
```

addInactiveQuery()

- Adds a filter to return inactive records.

```
var incGr = new GlideRecord('incident');
incGr.addInactiveQuery();
incGr.query();
if (incGr.next()) {
   gs.info(incGr.getDisplayValue('number'));
}
```

addNotNullQuery(String fieldName)

- Adds a filter to return records where the specified field is not `null`.

```
var incGr = new GlideRecord('incident');
incGr.addNotNullQuery("short_description");
incGr.query();
if (incGr.next()) {
   gs.info(incGr.getDisplayValue('number'));
}
```

addNullQuery(String fieldName)

- Adds a filter to return records where the specified field is `null`.

```
var incGr = new GlideRecord('incident');
incGr.addNullQuery("assigned_to");
incGr.query();
if (incGr.next()) {
   gs.info(incGr.getDisplayValue('number'));
}
```

UNION: Combining Results from Two Queries

SELECT title FROM Books WHERE genre = 'Science Fiction' UNION SELECT title FROM Books WHERE year > 2000;

- You want to list books that are either Science Fiction or were published after the year 2000.
- SELECT title FROM Books WHERE genre = 'Science Fiction': Finds titles of Science Fiction books.
- UNION: Combines results from both queries.
- SELECT title FROM Books WHERE year > 2000: Finds titles of books published after 2000.

Subqueries: Finding Books by Popular Authors

SELECT title FROM Books WHERE author_id IN (SELECT id FROM Authors WHERE popularity > 8);

- You want to find books written by very popular authors (popularity > 8).
- SELECT title FROM Books WHERE author_id IN: Tells the librarian to find books written by certain authors.
- (SELECT id FROM Authors WHERE popularity > 8): Finds authors with a popularity rating greater than 8.

addEncodedQuery(String query, Boolean enforceFieldACL)

- Adds an encoded query to other queries that may have been set.
- If there are multiple conditions in query, the conditions are ANDed.
- To use ORs or create technically complex queries, use encoded queries.

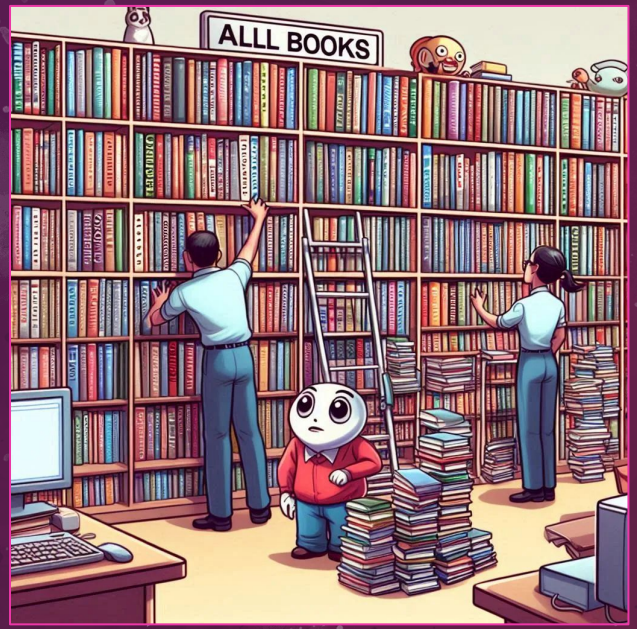
```
var incGr = new GlideRecord('incident');
incGr.addEncodedQuery("active=true");
incGr.query();
if (incGr.next()) {
   gs.info(incGr.getDisplayValue('cmdb_ci'));
}
```

getEncodedQuery()

- Retrieves the query condition of the current result set as an encoded query string.

```
var incGr = new GlideRecord('incident');
incGr.addActiveQuery();
incGr.query();
gs.info(incGr.getEncodedQuery());
if (incGr.next()) {
   gs.info(incGr.caller_id.location.name);
}
```

ORDER BY: Sorting Books



SELECT title, author FROM Books ORDER BY title ASC;

- You want to list all books sorted by title in ascending order.
- SELECT title, author: Tells the librarian you want the titles and authors.
- FROM Books: Specifies the section.
- ORDER BY title ASC: Orders the books by title in ascending (A-Z) order.

orderBy(String fieldName)

- Specifies a field name, or path to an attribute within a dynamic attribute store, to use to order the query set.
- To order by multiple fields, call this method multiple times with different field values.

```
var incGr = new GlideRecord('incident');
incGr.orderBy('number');
incGr.setLimit(5);
incGr.query();
while (incGr.next()) {
    gs.info(incGr.getDisplayValue());
}
```

orderByDesc(String fieldName)

- Specifies the field, or an attribute in a dynamic attribute store, to use to order the query set in descending order.

```
var incGr = new GlideRecord('incident');
incGr.orderByDesc('number');
incGr.setLimit(5);
incGr.query();
while (incGr.next()) {
    gs.info(incGr.getDisplayValue());
}
```

getLink(Boolean noStack)

- Retrieves the link for the current record.

```
var incGr = new GlideRecord('incident');
incGr.orderBy('number');
incGr.setLimit(1);
incGr.query();
while (incGr.next()) {
   gs.info(incGr.getLink(true));
}
```

get(Object name, Object value)

- Returns the specified record in the current GlideRecord object.
- This method accepts either one or two parameters.
- If only a single parameter is passed in, the method assumes that it is the sys_id of the desired record.
  - If not found, it then tries to match the value against the display value.
- If two parameters are passed in, the first is the name of the column within the GlideRecord to search.
  - The second is the value to search for.
- If multiple records are found, use next() to access the additional records.

get(Object name, Object value)

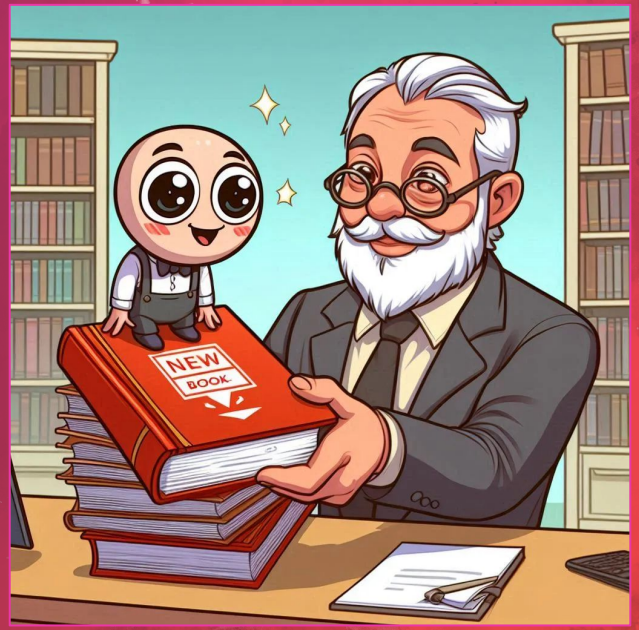
```
var incGr = new GlideRecord('incident');
incGr.get('1682fd87378c1300023d57d543990e2e');
gs.info(incGr.getDisplayValue('number'));
```

```
var incGr = new GlideRecord('incident');
incGr.get('number',"Demo_INC0010033");
gs.info(incGr.getDisplayValue('short_description'));
```

INSERT Query: Adding a New Book

```
INSERT INTO Books (title, author, genre, year)
VALUES ('Dune', 'Frank Herbert', 'Science
Fiction', 1965);
```

- You want to add a new book to the library.
- INSERT INTO: Instructs the librarian to add a new book to the Books section.
- VALUES: Provides the details of the new book (Dune, Frank Herbert, Science Fiction, 1965).



UPDATE Query: Updating Book Information

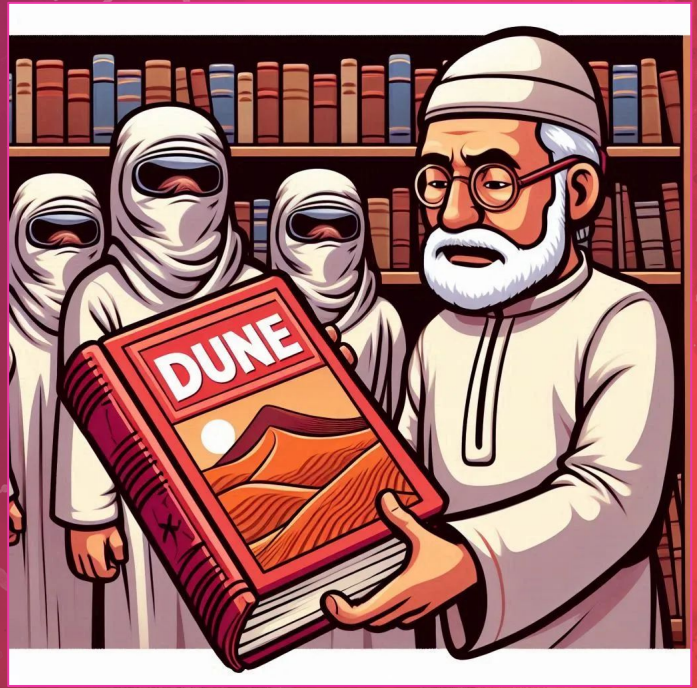
UPDATE Books SET year = 1966 WHERE title = 'Dune';

- You realize the publication year of "Dune" is incorrect and want to update it.
- UPDATE: Tells the librarian to change some details of a book.
- SET: Specifies the new information (year 1966).
- WHERE: Identifies which book to update (title 'Dune').

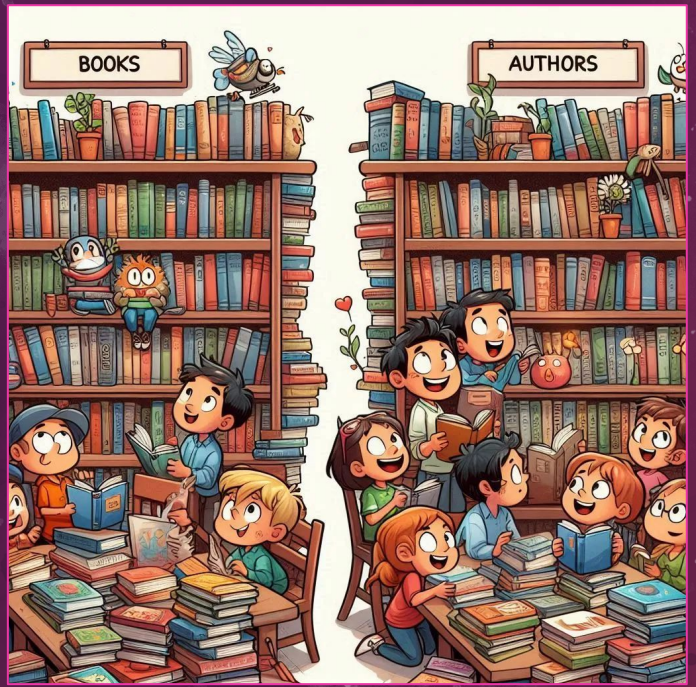
DELETE Query: Removing a Book

```
DELETE FROM Books WHERE title = 'Dune';
```

- You want to remove the book "Dune" from the library.
- DELETE FROM: Instructs the librarian to remove a book from the Books section.
- WHERE: Specifies which book to remove (title 'Dune').



JOIN Query: Combining Information from Different Shelves



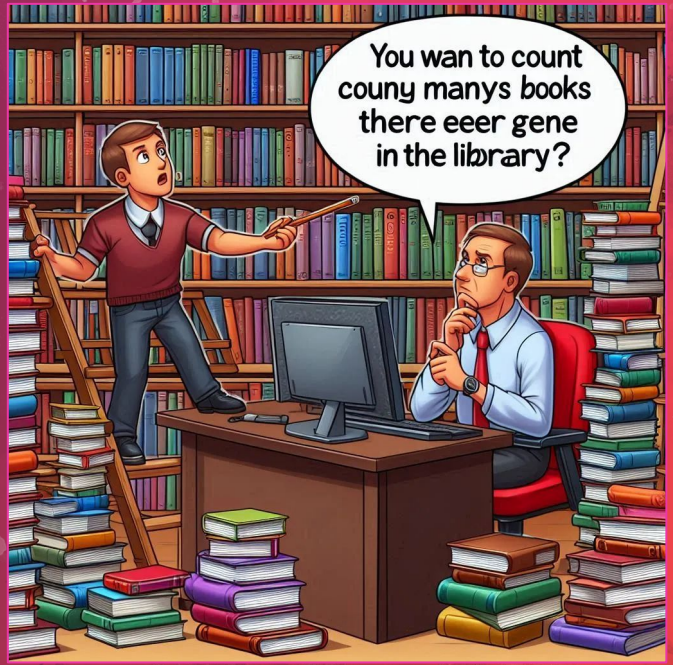
SELECT Books.title, Authors.name FROM Books JOIN Authors ON Books.author_id = Authors.id;

- You want to list all books along with their authors' names by looking at both the Books and Authors shelves.
- JOIN: Asks the librarian to combine information from two sections (Books and Authors).
- ON: Specifies how the information should be matched (Books.author_id = Authors.id).

SELECT with Aggregation: Counting Books

SELECT genre, COUNT (*) FROM Books GROUP BY genre.

- You want to count how many books there are in each genre in the library.
- SELECT genre, COUNT(*): Tell the librarian you want the genre and the number of books in each genre.
- FROM Books: Specifies you're looking in the Books section.
- GROUP BY genre: Group the books by genre to count them.



EQ

- A complex query is like a story told in SQL; If it's not well-written, it's hard to follow and understand.
- A good SQL query can solve many problems, but a bad one can cause endless headaches.

References

- Mastering SQL
- Server-side Scripting
- GlideRecord
- GlideRecord - Global
- GlideElement
- GlideQueryCondition