

# ServiceNow 201:

## *4 - GlideRecord*



# Course Outline

---

1 Course Introduction

2 Development Overview

3 Scripting Locations

4 GlideRecord

5 GlideSystem

6 GlideForm & GlideUser

7 GlideAjax

8 Exploring Other APIs

9 Becoming A Scripting Master

10 Creating A Custom App

# Section Outline

---

1 GlideRecord Introduction

2 Show Me The Code!

3 Concept: Dot-Walking

4 GlideRecord API Diagram

5 Common GlideRecord Methods

6 Stages Of A GlideRecord

7 Walking Through CRUD

8 GlideRecord Demo

9 GlideRecordSecure

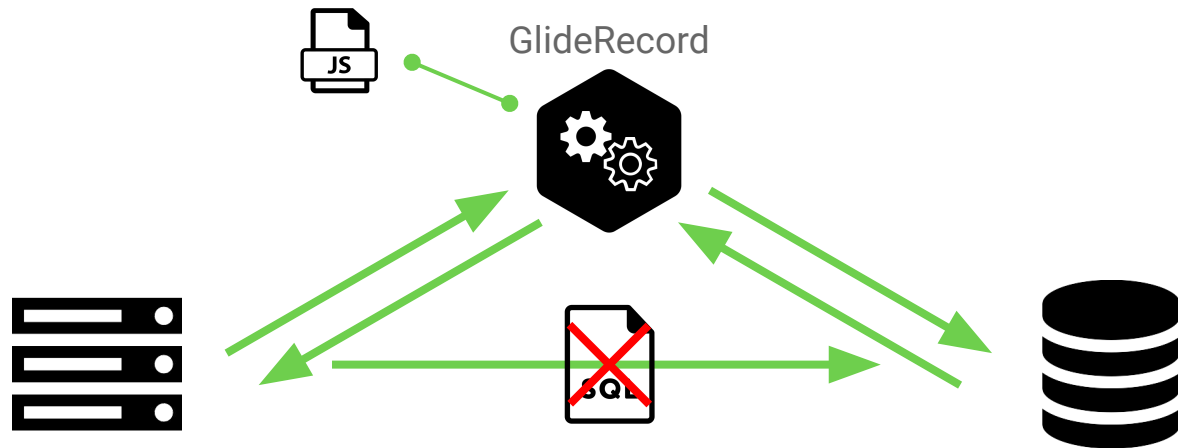
10 GlideAggregate

11 Where Can I Use This?

12 Section Recap

# Application Storage

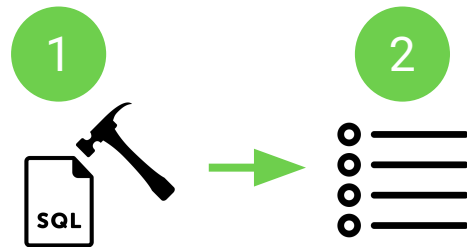
---



# GlideRecord Introduction



- Most common API
- Server side
- Used for database operations (CRUD)
- Generates SQL for you
- 2 stages
  - Building a query
  - Process records



C<sub>reate</sub>

R<sub>ead</sub>

U<sub>pdate</sub>









D<sub>ele</sub>



# Show Me The Code!

- Print a list of all priority 1 incidents to the screen

```
1 var incidentGR = new GlideRecord('incident');
2 incidentGR.addQuery('priority', 1);
3 incidentGR.query();
4 while(incidentGR.next()) {
5     gs.print(incidentGR.number);
6 }
```

<input type="checkbox"/>		 Created ▼	 Number	 Short description	 Category	 Opened by
<input type="checkbox"/>		<input type="text" value="Search"/>	<input type="text" value="Search"/>	<input type="text" value="Search"/>	<input type="text" value="Search"/>	<input type="text" value="Search"/>
<input type="checkbox"/>		<a href="#">2017-04-17 16:33:04</a>	<a href="#">INC0020156</a>	Help! The Internet Is Broken	Network	 <a href="#">Mark Miller</a>
<input type="checkbox"/>		<a href="#">2016-01-13 07:33:54</a>	<a href="#">INC0020009</a>	HELP!!!	Software	 <a href="#">System Administrator</a>



Best Practice: Use meaningful variable names with context

# GlideRecord By Analogy: Grocery Shopping

---

1. Go to a grocery store
2. Grab a shopping cart
3. Place groceries in the shopping cart
4. Checkout at cashier



Table



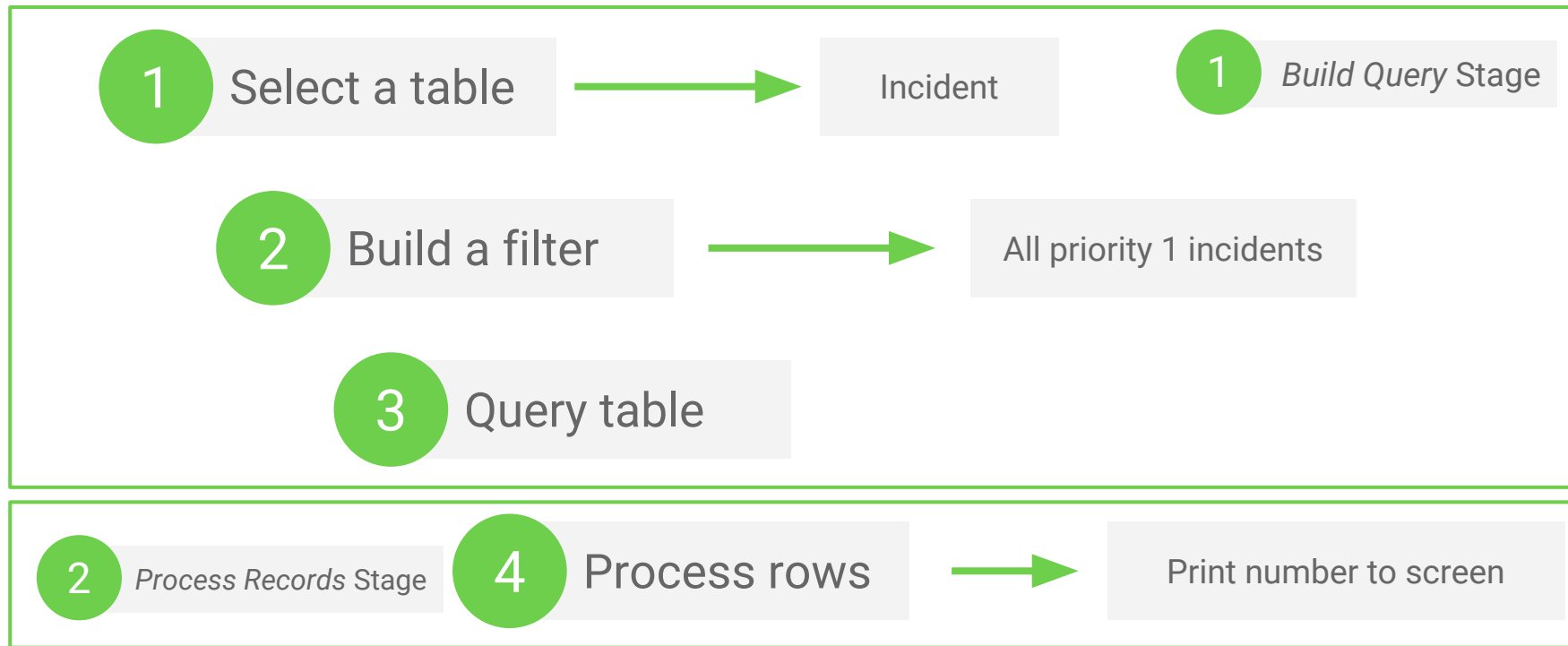
GlideRecord

Records



Action

# General GlideRecord Steps

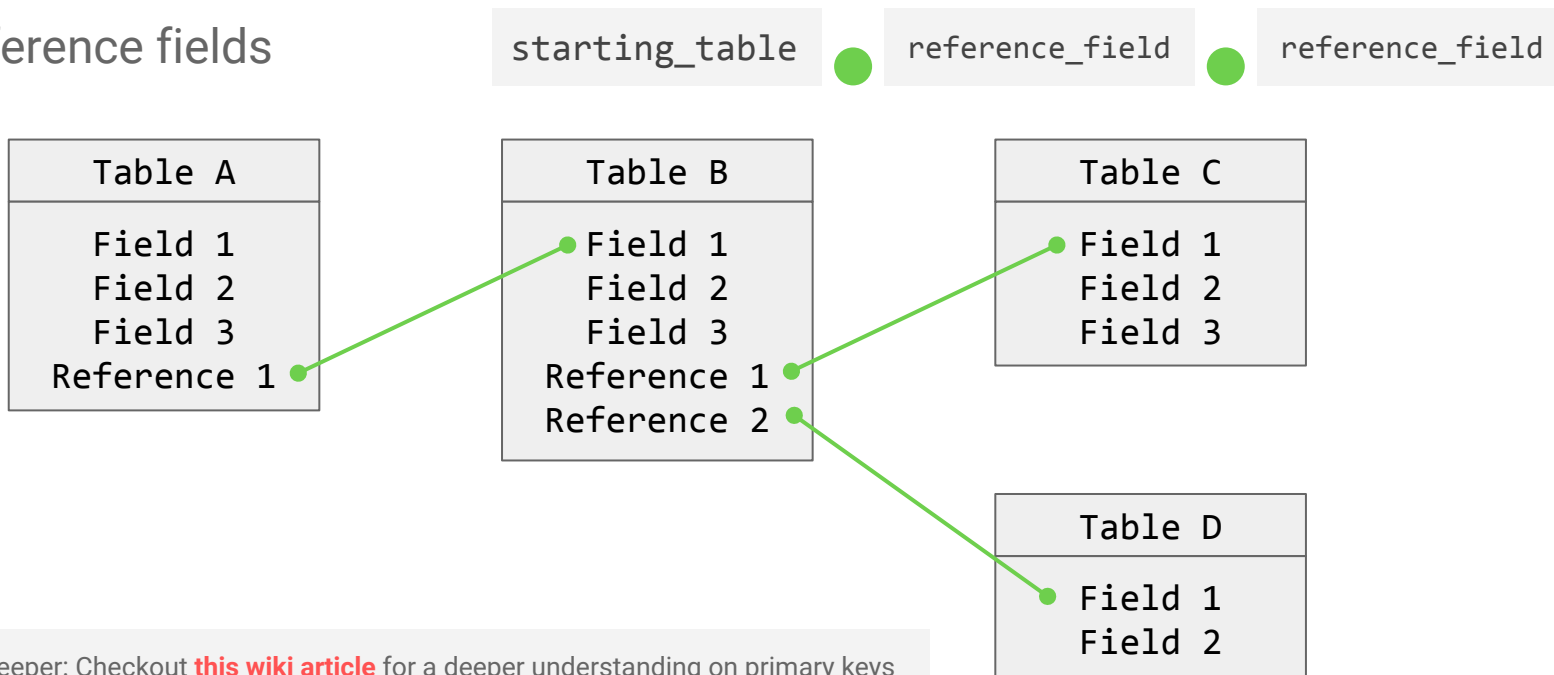


Note: Any ServiceNow table may be queried with GlideRecord



# Concept: Dot-Walking

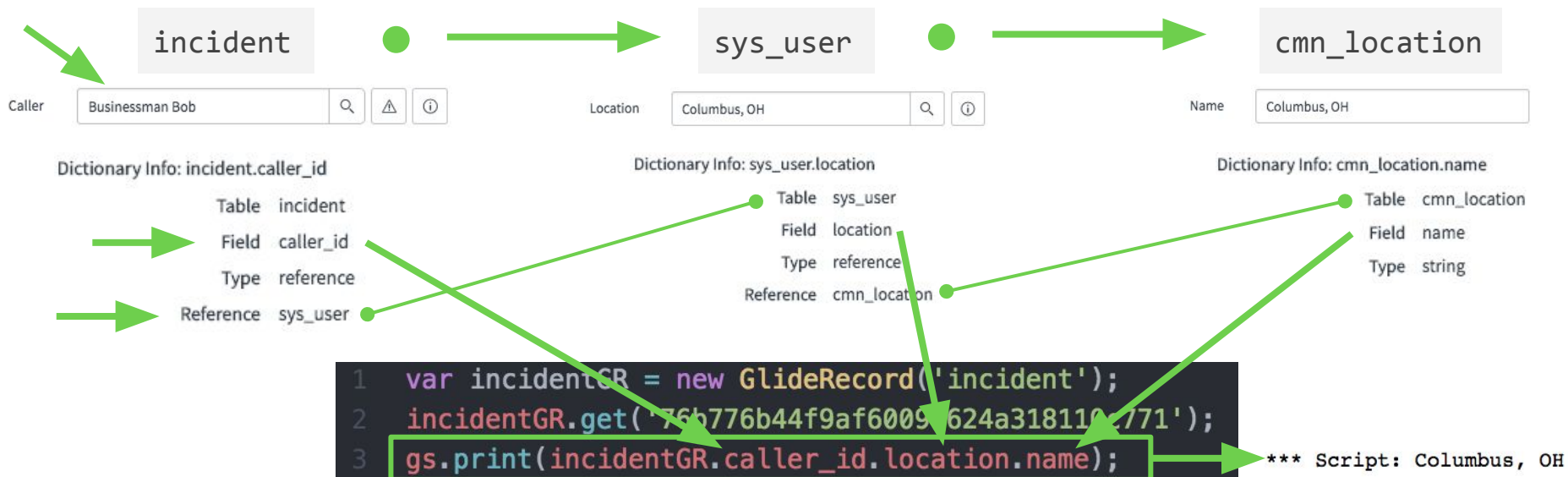
- Relational database
- Reference fields



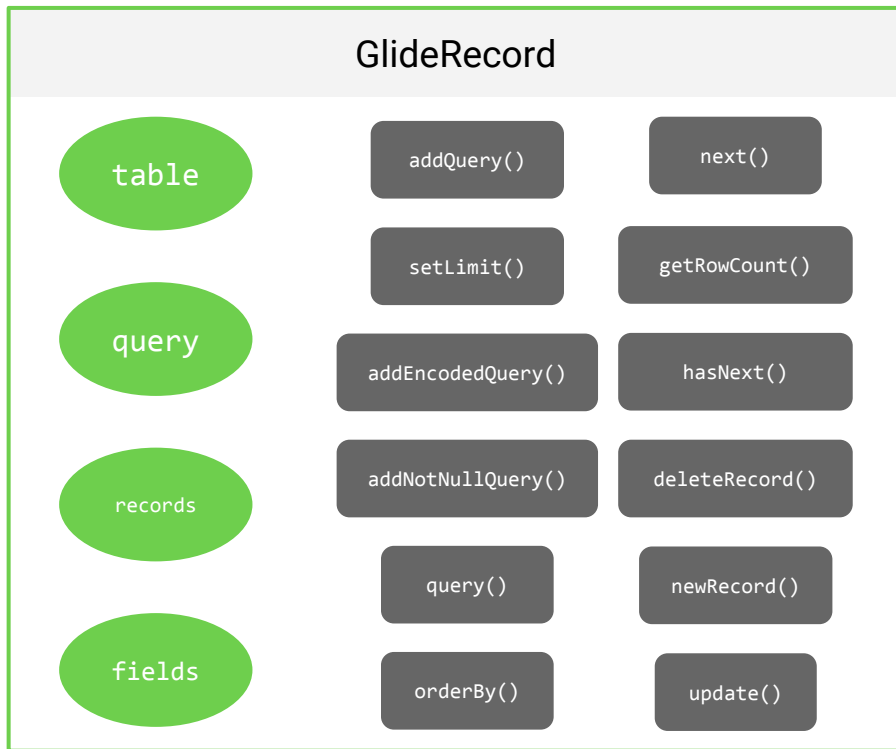
Dive Deeper: Checkout [this wiki article](#) for a deeper understanding on primary keys

# Example: Dot-Walking

- Example:
  - For a specific incident, you would like to find the location of the caller

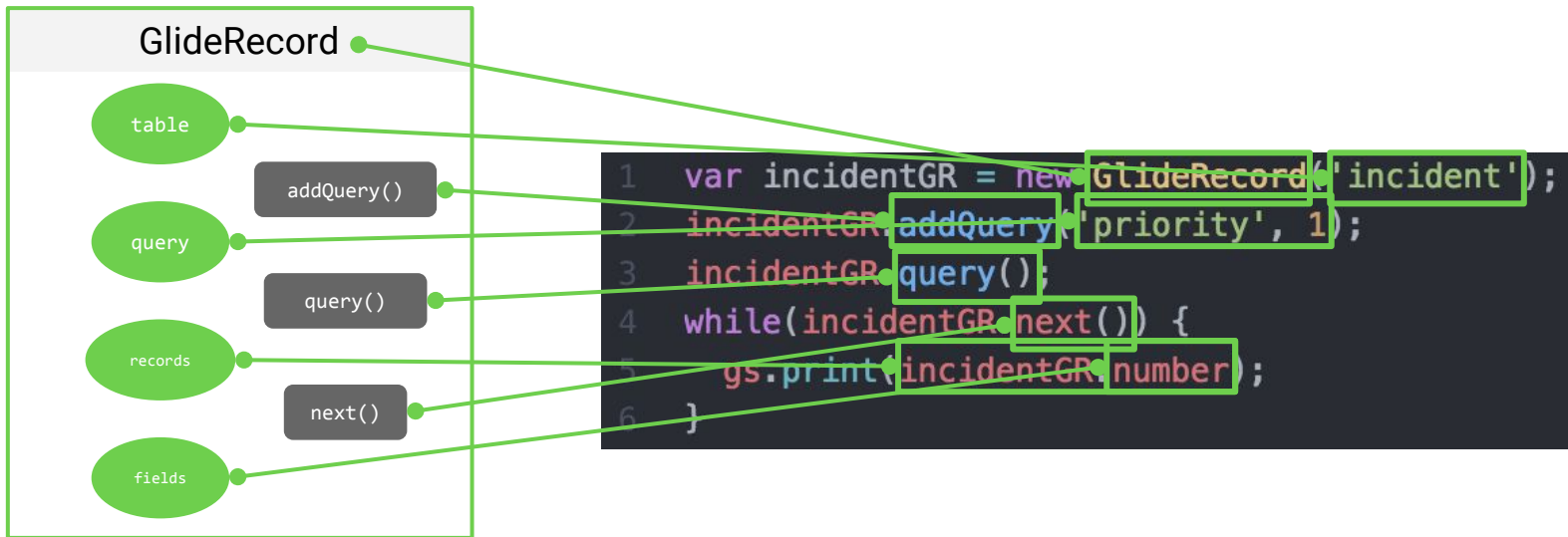


# GlideRecord API Diagram



Note: Ovals represent properties or topics associated with API, while rounded rectangles represent methods

# GlideRecord API Mapping



# Common GlideRecord Methods

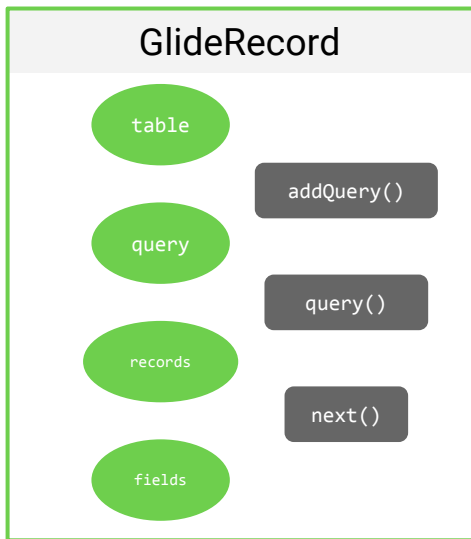
---

- `query()`
- `newRecord()`
- `insert()`
- `update()`
- `deleteRecord()`
- `addQuery()`
- `addEncodedQuery()`
- `hasNext()`
- `next()`
- `get()`
- `orderBy()`
- `orderByDesc()`
- `canCreate()`
- `canWrite()`
- `canRead()`
- `canDelete()`

# GlideRecord Stage 1: Building Query

1

Build Query Stage



```
GlideRecord = {  
  table: 'incident',  
  query: 'priority=1',  
  ...  
};
```



```
SELECT *  
FROM GlideRecord.table  
WHERE GlideRecord.query
```



# GlideRecord Stage 1: Options To Build Queries

---

1

## Chain Methods

- `addQuery()`
- `addOrCondition()`
- `addNullQuery()`
- `addNotNullQuery()`
- `addActiveQuery()`
- `addInactiveQuery()`

2

## Encoded Query

- `addEncodedQuery()`

# GlideRecord Stage 1: Option 1 - Chain Methods

1

1

Chain Methods

Add GlideRecord methods onto the  
current GlideRecord object

```
1 var incidentGR = new GlideRecord('incident');
2 var orCond1 = incidentGR.addQuery('priority', '1');
3 orCond1.addOrCondition('priority', '2');
4 var orCond2 = incidentGR.addQuery('category', 'hardware');
5 orCond2.addOrCondition('category', 'software');
6 incidentGR.addQuery('sys_created_on', '>', '2017-01-01 12:00:00');
7 incidentGR.addNotNullQuery('short_description');
8 incidentGR.query();
```



# GlideRecord addQuery() Method

1

- Accepts 2 or 3 arguments

2

Arguments

```
1 var incidentGR = new GlideRecord('incident');  
2 incidentGR.addQuery('short_description', 'Test');
```

field\_name

field\_value

3

Arguments

```
1 var incidentGR = new GlideRecord('incident');  
2 incidentGR.addQuery('short_description', 'CONTAINS', 'Test');
```

field\_name

operator

field\_value

Numbers

=	>=	>
!=	<=	<

Strings

=	STARTSWITH	DOES NOT CONTAIN
!=	ENDSWITH	INSTANCEOF
IN	CONTAINS	
NOT IN		



Note: '=' is assumed if there are only 2 arguments

# GlideRecord Stage 1: Option 2 - Encoded Query

1

1

Build Query

2

Copy Query

The screenshot shows the GlideBuilder interface for creating a query. At the top, there are tabs for 'Incidents', 'New', 'for text', and a search bar. Below this, a filter bar shows the current query: 'All > Priority = 1 - Critical .or. Priority = 2 - High > Category = Hardware .or. Category = Software > Created > 2017-01-01 12:00:00 > Short description is not empty'. The main section is titled 'INCIDENT CONDITIONS' and contains a list of conditions. The conditions are grouped by logical operators: 'OR' for the first two conditions (Priority is 1 - Critical, Priority is 2 - High), 'AND' for the next two conditions (Category is Hardware, Category is Software), and 'AND' for the last two conditions (Created after 2017-01-01 12:00:00, Short description is not empty). Each condition has a dropdown for the field, a dropdown for the operator, and a dropdown for the value. There are also buttons for 'OR' and 'AND' to combine conditions.

00 > Short description is not empty

Open new window  
Copy URL  
Copy query

priority=1^ORpriority=2^category=hardware^ORcategory=software^sys\_created\_on>javascript:gs.dateGenerate('2017-01-01','12:00:00')^short\_descriptionISNOTEMPTY

```
1 var incidentGR = new GlideRecord('incident');
2 incidentGR.addEncodedQuery("priority=1^ORpriority=2^category=hardware^ORcategory=software^sys_created_on>javascript:gs.dateGenerate('2017-01-01','12:00:00')^short_descriptionISNOTEMPTY");
3 incidentGR.query();
```

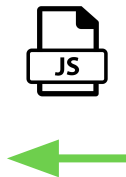
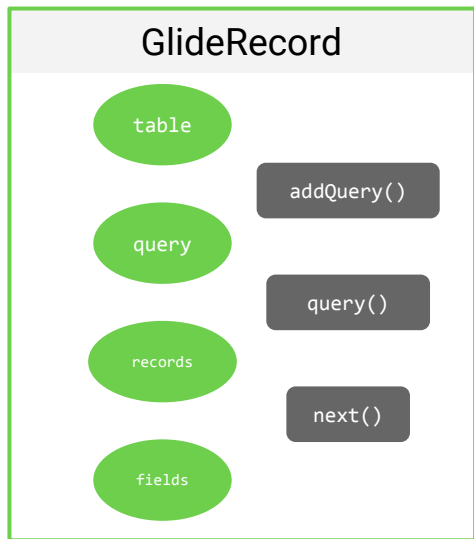
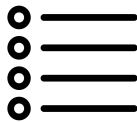
3

Add to addEncodedQuery()

# GlideRecord Stage 2: Process Records

2

Process Records Stage



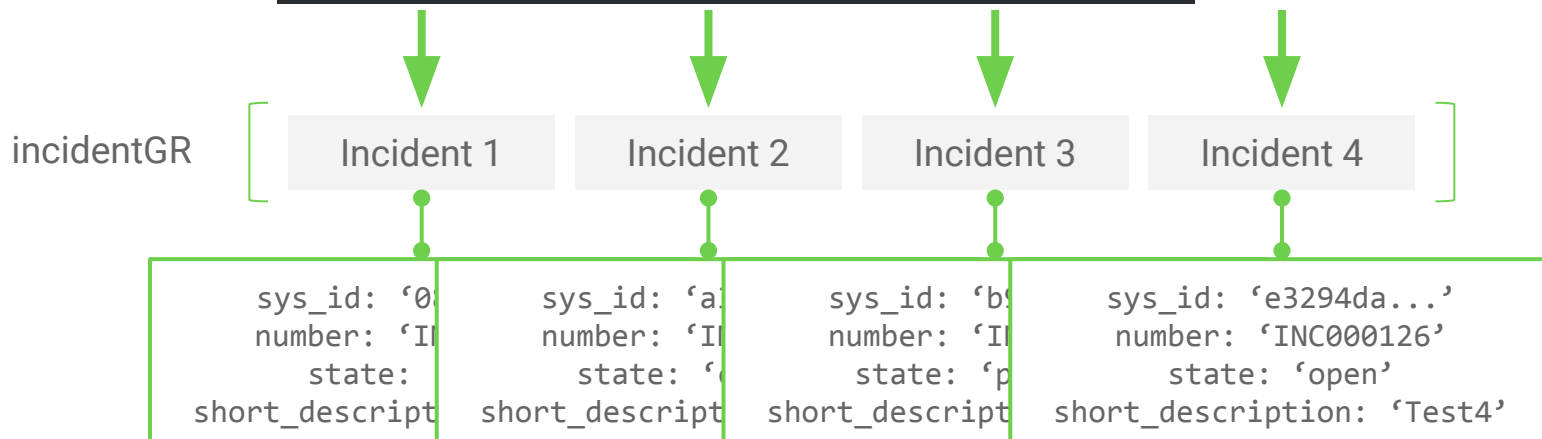
```
{  
  number: 'INC00001',  
  short_description: 'Test',  
  ...  
}
```

	A	B	C	D	E
1	number	priority	urgency	impact	...
2	INC00001	1	1	1	...
3	INC00002	2	1	3	...
4	INC00003	3	3	3	...

# GlideRecord next() Method & Iteration

2

```
1 var incidentGR = new GlideRecord('incident');
2 incidentGR.addQuery('priority', 1);
3 incidentGR.query();
4 while(incidentGR.next()) {
5     gs.print(incidentGR.number);
6 }
```



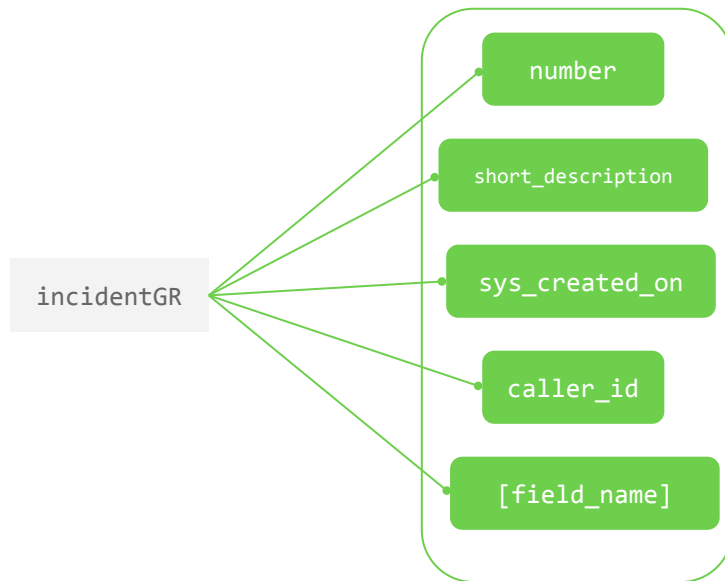
Dive Deeper: Checkout [this wiki article](#) and this [YouTube video](#) if you'd like to learn more on iterators

# Accessing A Record's Fields

2

- Once query() method is executed and stage 2 begins, all fields are just a dot away
- Fields become GlideRecord properties

```
1 var incidentGR = new GlideRecord('incident');
2 incidentGR.addQuery('priority', 1);
3 incidentGR.query();
4 while(incidentGR.next()) {
5     gs.print(incidentGR.number);
6 }
```

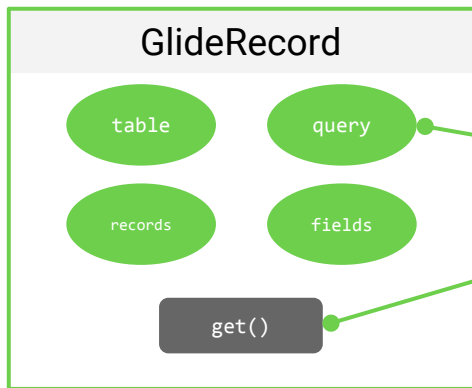


# GlideRecord get() Method

1

2

- Shortcut
- Only grabs 1 record
- Commonly used with record sys\_id



```
1 var gr = new GlideRecord('incident');  
2 gr.get('number', 'INC0000001');  
3 gs.print(gr.short_description);
```

# CRUD GlideRecord Mapping

---

`.insert()`

`.query()*`

`.update()`

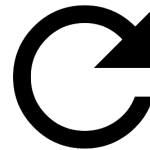
`.deleteRecord()`

**C**  
reate

**R**  
ead

**U**  
pdate

**D**  
elete



# CRUD - Create

---

1. Build GlideRecord
2. query()
3. newRecord()
4. Set field values
5. insert()

```
1 // Create
2 var incidentGR = new GlideRecord('incident');
3 incidentGR.query();
4 incidentGR.newRecord();
5 incidentGR.short_description = 'Test 123';
6 incidentGR.insert();
```



# CRUD - Read

---

1. Build GlideRecord
2. Add filter conditions (optional)
3. query()
4. next()
5. Print or copy variables

```
1 // Read
2 var incidentGR = new GlideRecord('incident');
3 incidentGR.addQuery('priority', '1');
4 incidentGR.query();
5 while(incidentGR.next()) {
6     // Read record(s)
7     gs.print(incidentGR.number);
8 }
```

# CRUD - Update

---

1. Build GlideRecord
2. Add filter conditions (optional)
3. query()
4. next()
5. Set field values
6. update()

```
1 // Update
2 var incidentGR = new GlideRecord('incident');
3 incidentGR.addQuery('priority', '1');
4 incidentGR.query();
5 while(incidentGR.next()) {
6     // Update records
7     incidentGR.priority = '2';
8     incidentGR.update();
9 }
```

# CRUD - Delete

---

1. Build GlideRecord
2. Add filter conditions (optional)
3. query()
4. next()
5. deleteRecord()

```
1 // Delete
2 var incidentGR = new GlideRecord('incident');
3 incidentGR.addQuery('number', 'INC0000001');
4 incidentGR.query();
5 while(incidentGR.next()) {
6     // Delete record
7     incidentGR.deleteRecord();
8 }
```

# GlideRecordSecure

---

- GlideRecordSecure class is inherited from GlideRecord
  - Has all of the same methods
  - Performs ACL checking
- Used to secure Script Includes
- Replaces `canWrite()`, `canRead()`, `canUpdate()`, `canDelete()` GlideRecord methods



Dive Deeper: Watch [episode 15 of TechNow](#) to learn more about GlideRecordSecure

# GlideRecord Versus GlideRecordSecure

```
1 // GlideRecord
2 // reading records
3 var incidentGR = new GlideRecord('incident');
4 incidentGR.query();
5 while(incidentGR.next()) {
6     if(incidentGR.canRead()) {
7         // read records
8     }
9 }
10
11 // creating records
12 var incidentGR = new GlideRecord('incident');
13 if(incidentGR.canWrite()) {
14     // create record
15 }
16
17 // updating records
18 var incidentGR = new GlideRecord('incident');
19 incidentGR.query();
20 while(incidentGR.next()) {
21     if(incidentGR.canUpdate()) {
22         // update records
23     }
24 }
25
26 // deleting records
27 var incidentGR = new GlideRecord('incident');
28 if(incidentGR.canDelete()) {
29     // delete record
30 }
```

VS

```
34 // GlideRecordSecure
35 var incidentGRS = new GlideRecordSecure('incident');
36 incidentGRS.query();
37 while(incidentGRS.next()) {
38     // do something
39 }
```

# GlideAggregate

---

- Extension of GlideRecord class
- Used when performing aggregate queries (count, min, max, sum, avg)
  - Reports
  - Calculations
- Only works on number fields

```
1  var count = new GlideAggregate('incident');
2  count.addAggregate('COUNT');
3  count.query();
4  var incidents = 0;
5  if (count.next()) {
6      incidents = count.getAggregate('COUNT');
7  }
```

# Where Can I Use This?

## Client-Side

GlideForm



Client Scripts



UI Actions

GlideUser



UI Policies

GlideAjax



Service Portal

## Server-Side

GlideRecord



Business Rules



UI Actions

GlideSystem



Script Includes

GlideDateTime



Scheduled Jobs



Service Portal



Web Services



Workflows



Note: GlideRecord on the client side is not recommended, thus will not be discussed in this course

# Section Recap

---

- Use GlideRecord when dealing with database operations
- CRUD stands for Create, Read, Update, Delete
- GlideRecord is a server-side API
- There are 2 stages to the GlideRecord API
  - 1) *Query building* stage
  - 2) *Process records* stage
- Use the `next()` method to iterate over the returned records
- Use the `get()` method to retrieve a specific record using a unique field, such as the `sys_id`





## Section Recap (cont.)

---

- Use GlideAggregate over GlideRecord when dealing with aggregates like *counts*
- Log records before deleting
- Avoid using GlideRecord in Client Scripts
- Use GlideRecordSecure where appropriate

