Apex Basic & DataBase

**Salesforce Indexes**

Практика  
 IndexedScratch  
 Object – AirPort  
 Indexed field in ApexBasic project  
 поле на котором можно показать iso\_region (не индексировано)  
 индекс на двух полях airport\_id\_\_c и local\_code\_\_c

Indexes <https://www.youtube.com/watch?v=Wq8q8ZgC89A&ab_channel=SteveTechArc>  
QueryPlanTool <https://www.youtube.com/watch?v=RQAGl_mRT2E&ab_channel=SteveTechArc>  
  
SF Doc  
**Indexes** https://developer.salesforce.com/docs/atlas.en-us.salesforce\_large\_data\_volumes\_bp.meta/salesforce\_large\_data\_volumes\_bp/ldv\_deployments\_infrastructure\_indexes.htm  
**Custom Indexes** https://developer.salesforce.com/docs/atlas.en-us.api\_meta.meta/api\_meta/meta\_customindex.htm  
**Index for BigObject** <https://developer.salesforce.com/docs/atlas.en-us.api_meta.meta/api_meta/meta_index.htm>  
**QueryOptimizer** <https://developer.salesforce.com/docs/atlas.en-us.salesforce_large_data_volumes_bp.meta/salesforce_large_data_volumes_bp/ldv_deployments_infrastructure_salesforce_query_optimizer.htm>  
  
**Default indexes from SF**  
RecordTypeId

Division

CreatedDate

Systemmodstamp (LastModifiedDate)

Name

Email (for contacts and leads)

Foreign key relationships (lookups and master-detail)

The unique Salesforce record ID, which is the primary key for each object

Salesforce also supports custom indexes on custom fields, **except** for **multi-select picklists,** **text areas (long), text areas (rich), non-deterministic formula fields (win13r), and encrypted text fields.**

## Index Tables

The Salesforce multitenant architecture makes the underlying data table for custom fields unsuitable for indexing. To overcome this limitation, the platform creates an index table that contains a copy of the data, along with information about the data types.

The platform builds a standard database index on this index table. The index table places upper limits on the number of records that an indexed search can effectively return.

By default, the index tables don’t include records that are null (records with empty values). You can work with Salesforce Customer Support to create custom indexes that include null rows. Even if you already have custom indexes on your custom fields, you must explicitly enable and rebuild them to get the empty-value rows indexed.

## Standard and Custom Indexed Fields

The query optimizer **maintains a table containing statistics about the distribution of data in each index**. **It uses this table to perform pre-queries to determine whether using the index can speed up the query**.

**Условия использования Индексов со стороны оптимизатора**

**Standard Indexed Fields**

Used if the filter matches **less than 30% of the first million records** and **less than 15% of additional records**, up to a **maximum of one million records**.

**Custom Indexed Fields**

Used if the filter matches **less than 10% of the total records**, up to a **maximum of 333,333 records**.

If the criteria for an indexed field aren’t met, only that index is excluded from the query. If they are in the WHERE clause and meet the thresholds for records, other indexes are sometimes used.

The query optimizer uses similar considerations to determine whether to use indexes when the WHERE clause contains AND, OR, or LIKE.

* For AND, the query optimizer uses the indexes unless one of them returns more than 20% of the object’s records or 666,666 total records.
* For OR, the query optimizer uses the indexes unless they all return more than 10% of the object’s records or 333,333 total records.  
  !!! All fields in the OR clause must be indexed for any index to be used.   
    
  For LIKE, the query optimizer doesn’t use its internal statistics table. Instead, **it samples up to 100,000 records of actual data to decide whether to use the custom index.**

Custom indexes can be created on deterministic formula fields. Because some values vary over time or change when a transaction updates a related entity, the platform can’t index non-deterministic formulas.

Here are examples of things that make formula fields non-deterministic.

Non-deterministic formula fields can:

* Reference other entities (like fields accessible through lookup fields)
* Include other formula fields that span over other entities
* Use dynamic date and time functions (for example, TODAY and NOW)

These formula fields are also considered non-deterministic

* Owner, autonumber, divisions, or audit fields (except for CreatedDate and CreatedByID fields
  + References to fields that Lightning Platform can’t index
  + Multi-select picklists
  + Currency fields in a multicurrency organization
  + Long text area fields
  + Binary fields (blob, file, or encrypted text)
* Standard fields with special functionalities
  + Opportunity: Amount, TotalOpportunityQuantity, ExpectedRevenue, IsClosed, IsWon
  + Case: ClosedDate, IsClosed
  + Product: ProductFamily, IsActive, IsArchived
  + Solution: Status
  + Lead: Status
  + Activity: Subject, TaskStatus, TaskPriority

Cross-object indexes are typically used if specified using the cross-object notation, as they are in the following example. CrossObject1\_\_r.CrossObject2\_\_r.IndexedField\_\_c

You can use this approach to replace formula fields that can’t be custom-indexed because they reference other objects. As long as the referenced field is indexed, the cross-object notation can have multiple levels.

## Two-Column Custom Indexes

Two-column indexes are subject to the same restrictions as single-column indexes, with one exception. Two-column indexes can have nulls in the second column, whereas single-column indexes can’t unless Salesforce Customer Support explicitly enabled the option to include nulls.

**Maximum number of custom indexes per entity**

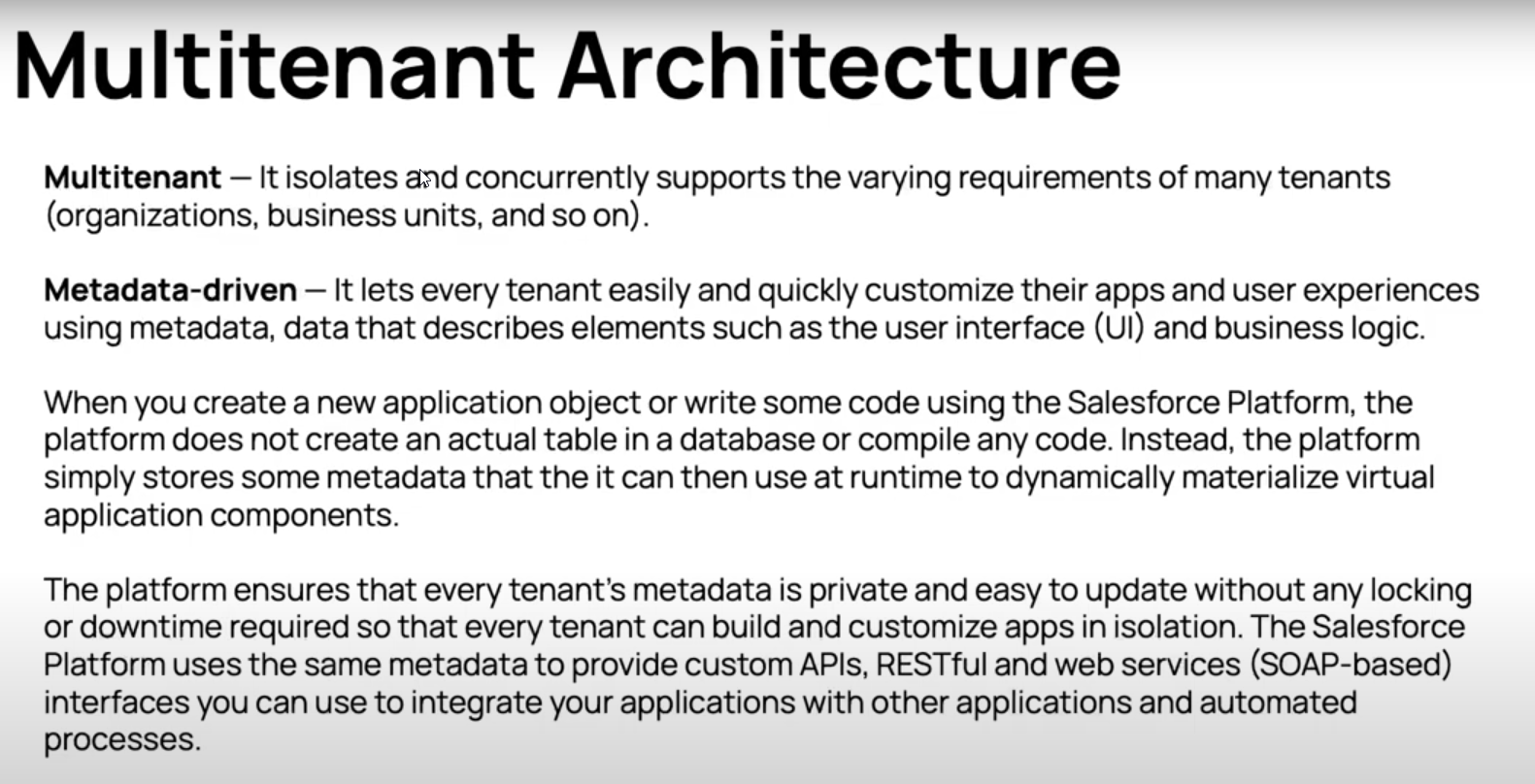
https://help.salesforce.com/s/articleView?id=000386304&type=1

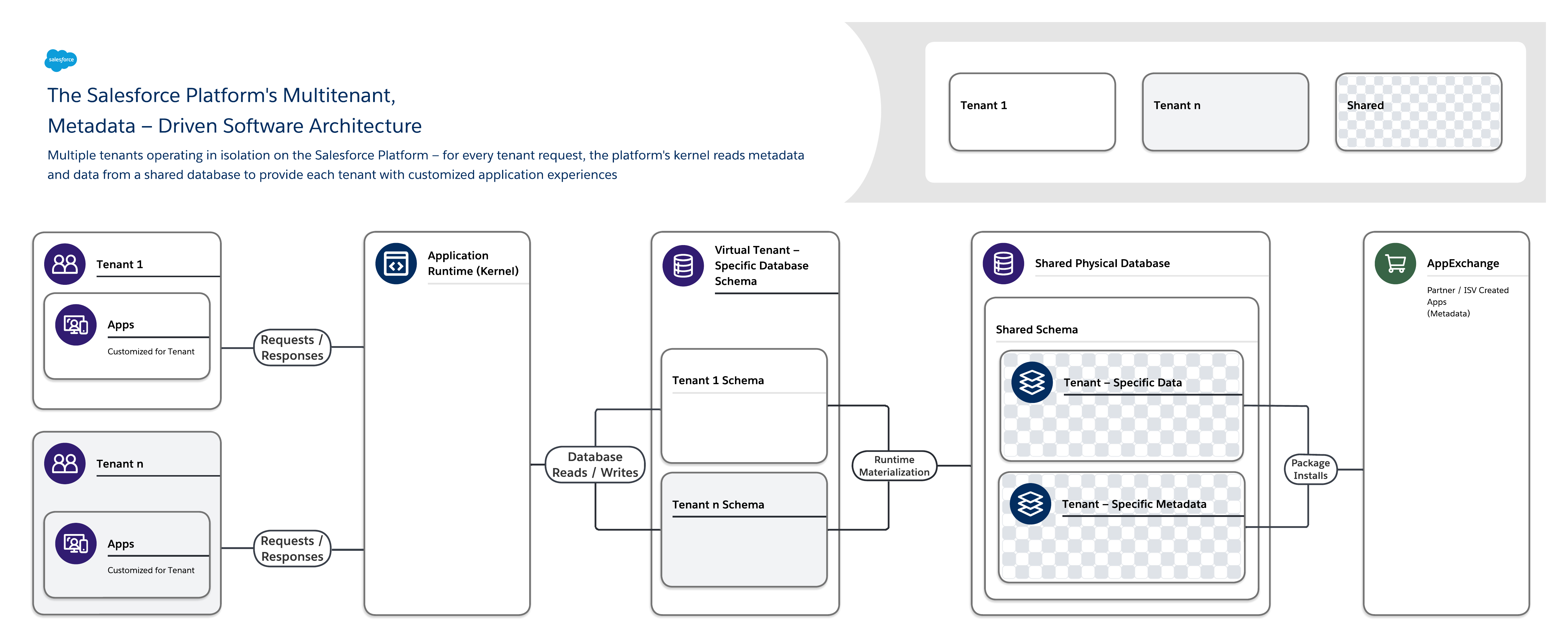
The default limit for the maximum number of custom indexes per entity (object) within an Org is 25.   
Custom indexes can be used as a tool to assist with slowness or time-outs in certain situations. There are three ways custom indexes are created. Either:  
  
A) By marking a field as "External" or "Unique". NOTE: This has its own limit as well which is defaulted at 7.  
  
B) Some indexes are created automatically when our systems determine a field will benefit from it.  
  
C) By Salesforce Support, as part of a performance case. (Ex: SOQL queries or Analytic Reports timing out)

* Please note that custom indexes are not the only tool that should and/or will be used in any performance case.
* Custom indexes, when used improperly, can actually slow query results.
* Please reach out to Support if you have any questions or concerns on indexing or a performance issue. When creating a case, be specific and mention if the performance issue exists within a SOQL query, a report, etc (with links and [login access](https://help.salesforce.com/s/articleView?language=en_US&type=1&id=How-to-Grant-Login-Access-to-a-Salesforce-com-Support-Analyst-1327108337089)) so we can route the case to the appropriate team for assistance.

# Make SOQL query selective

https://help.salesforce.com/s/articleView?id=000385218&type=1

https://architect.salesforce.com/fundamentals/platform-multitenant-architecture



### Indexes

The platform automatically indexes various types of fields to deliver scalable performance.

Traditional database systems rely on native database indexes to quickly locate specific rows in a database table that have fields matching a specific condition. However, it is not practical to create native database indexes for the flex columns of MT\_Data because the platform uses a single flex column to store the data of many fields with varying structured data types. Instead, the platform manages an index of MT\_Data by synchronously copying field data marked for indexing to an appropriate column in an MT\_Indexes pivot table.

MT\_Indexes contains strongly typed, indexed columns such as StringValue, NumValue, and DateValue that the platform uses to locate field data of the corresponding data type. For example, the platform would copy a string value in an MT\_Data flex column to the StringValue field in MT\_Indexes, a date value to the DateValue field, and so on. The underlying indexes of MT\_Indexes are standard, non-unique database indexes. When an internal system query includes a search parameter that references a structured field in an object, the platform’s custom query optimizer uses MT\_Indexes to help optimize associated data access operations.

Note: The platform can handle searches across multiple languages because the system uses a case-folding algorithm that converts string values to a universal, case-insensitive format. The StringValue column of the MT\_Indexes table stores string values in this format. At runtime, the query optimizer automatically builds data access operations so that the optimized SQL statement filters on the corresponding case-folded StringValue, which in turn corresponds to the literal provided in the search request.

The platform lets you indicate when a field in an object must contain unique values (case-sensitive or case-insensitive). Considering the arrangement of MT\_Data and shared usage of the Value columns for field data, it is not practical to create unique database indexes for the object. (This situation is similar to the one discussed in the previous section for non-unique indexes.)

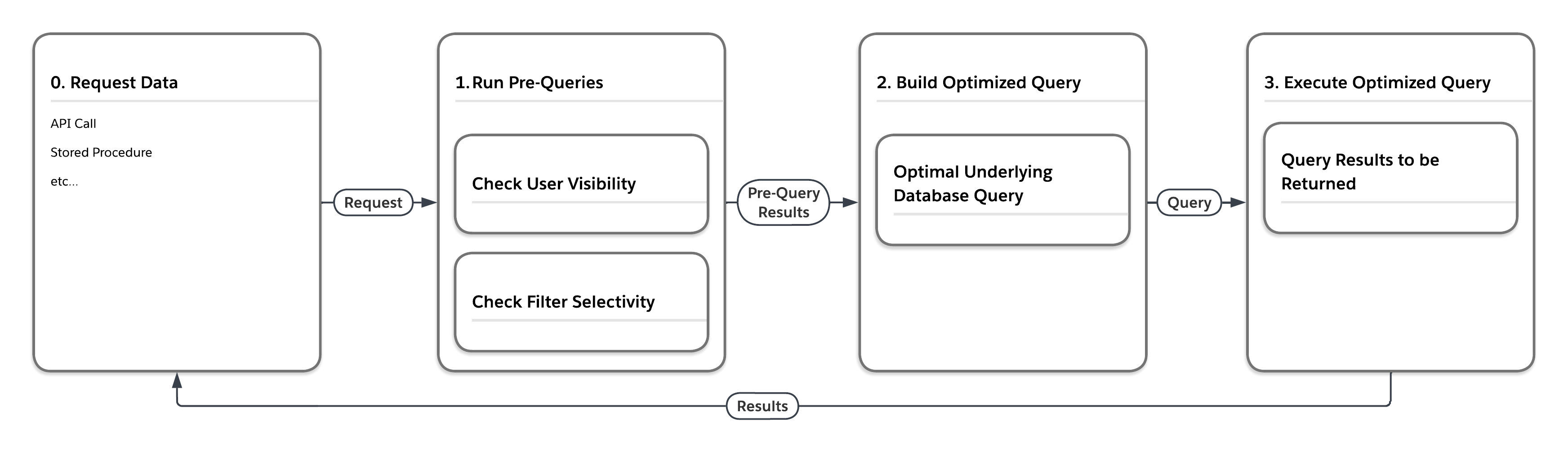
To support uniqueness for custom fields, the platform uses the MT\_Unique\_Indexes pivot table; this table is very similar to the MT\_Indexes table, except that the underlying native database indexes of MT\_Unique\_Indexes enforce uniqueness. When an application attempts to insert a duplicate value into a field that requires uniqueness, or an administrator attempts to enforce uniqueness on an existing field that contains duplicate values, the platform returns an appropriate error message to the application.

In rare circumstances, the platform’s external search engine (explained in [Searches](https://architect.salesforce.com/fundamentals/platform-multitenant-architecture" \l "Searches)) can become overloaded or otherwise unavailable, and may not be able to respond to a search request in a timely manner. Rather than returning a disappointing error to the end user, the platform falls back to a secondary search mechanism to furnish reasonable search results.

A fall-back search is implemented as a direct database query with search conditions that reference the Name field of target records. To optimize global object searches (searches that span tables) without having to execute potentially expensive union queries, the platform maintains a MT\_Fallback\_Indexes pivot table that records the Name of all records. Updates to MT\_Fallback\_Indexes happen synchronously as transactions modify records so that fall-back searches always have access to the most current database information.

The MT\_Name\_Denorm table is a lean data table that stores the ObjID and Name of each record in MT\_Data. When an application needs to provide a list of records involved in a parent/child relationship, the platform uses the MT\_Name\_Denorm table to execute a relatively simple query that retrieves the Name of each referenced record for display in the app, for example as part of a hyperlink.

---  
The flow diagram in the following figure illustrates what happens when the platform processes a request for data that is in one of the large heap tables such as MT\_Data.   
 - First, the platform executes “pre-queries” that consider the multitenant-aware statistics.   
 - Then, based on the results returned by the pre-queries, the service builds an optimal underlying database query for execution in the specific setting.

As the following table shows, the platform can execute the same query four different ways, depending on the user that submits the query and the selectivity of the query’s filter conditions.

|  |  |  |
| --- | --- | --- |
| Pre-Query Selectivity Measurements | | Write final database access query, forcing... |
| User | Filter |  |
| Low | Low | ... nested loops join, drive using view of rows that user can see |
| Low | High | ... use of index related to filter |
| High | Low | ... ordered hash join, drive using MT\_DATA |
| High | High | ... use of index related filter |

Example of metadata custom indexes  
in the project  
 force-app→main→default→ customindex → Airport\_\_c.local\_code\_\_c.indx  
<?xml version="1.0" encoding="UTF-8"?>

<CustomIndex xmlns="http://soap.sforce.com/2006/04/metadata">

    <allowNullValues>true</allowNullValues>

</CustomIndex>

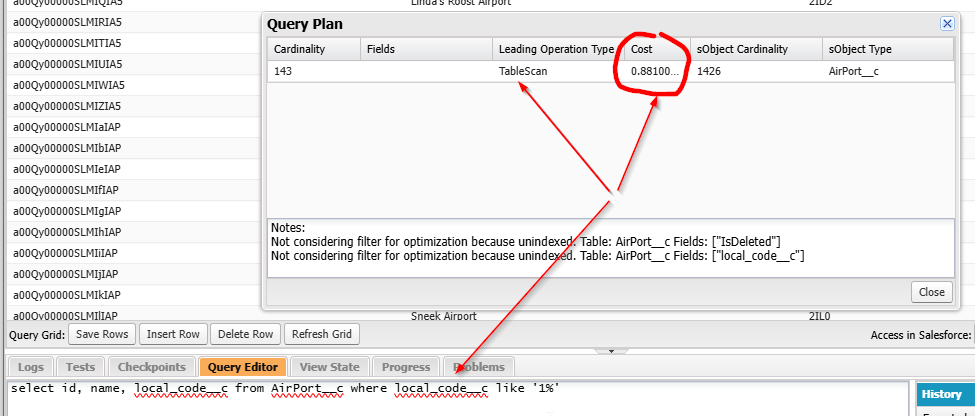
force-app→main→default→ customindex → Airport\_\_c.airport\_id\_\_c,local\_code\_\_c.indx

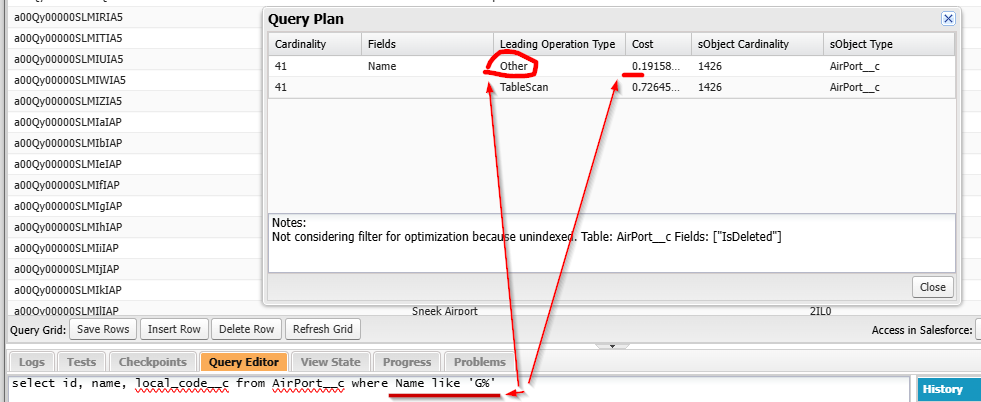
<?xml version="1.0" encoding="UTF-8"?>

<CustomIndex xmlns="http://soap.sforce.com/2006/04/metadata">

    <allowNullValues>false</allowNullValues>

</CustomIndex>

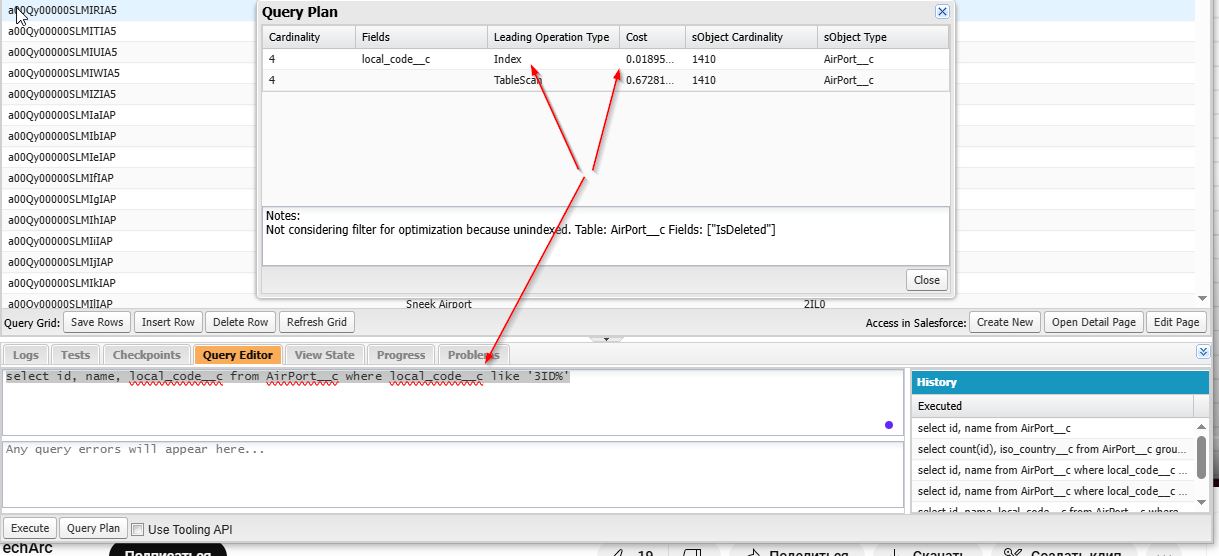
-----------  
  
Dev Console → Help → Preferences → Enable Query Plan  
  
Query Plan is tool which allow us to see QueryOptimazer work and statistics  
  
examples  
 **request by not indexed field**

запрос по не индексируемому полю как результат СФ использовало сканирование таблицы (проход по всем элементам таблицы) и оценочная стоимость 0.8810  
  
**request by Name (default indexed) field**

стоимость 0.19 по сравнению со сканом 0.72 и метод выбора уже не сканирование таблицы  
дальше если обьем выборки по первому милиону и последующий обьем удовлетворяет условиям то будет оптимизированная выборка

**create index for local\_code\_\_c field request by indexed field**

*select id, name, local\_code\_\_c from AirPort\_\_c where local\_code\_\_c like '3ID%'*

**