Export To SQL Known Limitations Guide

September 28, 2023

Table of Contents

Introduction

This document describes all of the known limitations, and the specifications for exporting-to-sql.

Export to MySQL

The following limitations exist for exporting data to MySQL:

- MySQL clients limit the table row size to a maximum of 65,535 bytes, excluding TEXT and BLOB fields. When we encounters a table where the sum of field sizes could exceed 65,535 bytes, it converts all the VARCHAR type custom Salesforce fields that aren't defined as keys or unique to TEXT type.
- MySQL does not support the Boolean data type (True / False). We will define the field as a **TINYINT(1)** data type, and then insert the integer value of **0** for Salesforce's boolean value of '**false**', and insert the integer value of **1** for Salesforce's boolean value of '**true**'.
- Row size limitations: In some cases where an object schema contains a large number of fields (the number of fields depends on their type, so there's no easy way to tell this ahead of time), an error relating to maximum row sizes may occur. This appears to be a known issue with MySQL.
  - It appears that for MySQL 5.7.16 (specifically), the following parameter settings can resolve the problem:
    - innodb_strict_mode = OFF
    - innodb_log_file_size = 536870912
    - innodb_default_row_format = dynamic
    - innodb_file_format = Barracuda
  - Alternatively, changing the DB Engine to MyISAM should remove this limitation. Note that AWS RDS does not allow changing the engine to MyISAM.
    - nnodb_page_size = 64k (AWS_RDS does not support changing this)
    - innodb_log_buffer_size = 32M
    - innodb_buffer_pool_size = 512M

Export to Oracle

The following limitations exist:

- Max field value length. When issuing multiple insert statements, Oracle prevents us from inserting values larger than 4000 bytes. Currently, such field values are trimmed to 4000 bytes. A comment is added to the SQL with the original field value.
- The max field length of field names in Oracle is 30 characters. In order to accommodate field names that are longer than 30 characters, the names are trimmed, and then concatenated with a running index number - from 1 to 999 - to the field name, in order to distinguish one trimmed field from another field. A comment indicating the original name (the Salesforce original field name) is added to the field. This lets the user map the original names to the trimmed ones.For example: *OptionsSendClientCredentialsInHeader* changes to *OptionsSendClientCredential001*.
- Duplicate index names. An index name that was used once can't be used again in Oracle. Similar to the above case of fields names, we concatenate a running index number - from 1 to 999 - to the name (after trimming the index name if it exceeds 30 characters). For example: *Name* changes to *Name001*.
- Boolean data type is not supported in Oracle. We will define the field as **NUMBER(1)** data type, and insert a value of **0** for Salesforce's boolean value '**false**', and insert a value of **1** for Salesforce's boolean value '**true**'.
- Oracle does not support a 'time-of-day' field type (such as: HH24:MI:SS). We will instead store the time in a **VARCHAR2(8)** field type. When saved in the format of **HH24:MI:SS**, the value consists of 8 characters, for example: 03:25:11. We convert Salesforce's time values to strings, and then store them in the field we've defined.

Export to PostgreSQL

The following limitations exist:

- In PostgreSQL, an index name that was used once, cannot be used again as both an index name and a table name. In order to overcome this limitation, we concatenate a running index number - from 1 to 999 - to the name. For example: **Name** changes to **Name001**, after the index number concatenation.

Export to MS SQL

The following limitations exist:

- MS SQL does not allow multiple NULL values, in columns that were defined as ‘Unique’. Whereas all the other SQL implementations that we support, do allow multiple NULL values. Therefore, we use a technique in which a unique, non-clustered index, is declared for the field post creation, in order to allow similar behaviour that's allowed on the other database types (other than MS SQL).
- MS SQL has a limit of 1000 lines per multiple insert statement. To work around this limit, we will split the records between several insert statements.
- In MS SQL, an index name that was used once, cannot be used again, neither as an index name nor as a table name. In order to overcome this limitation, we concatenate to the name, a running index number - from 001 to 999. For example: A field called **Name** is changed to **Name001**, after the index number concatenation.
- MS SQL does not support the Boolean data type (True / False). We will define the field as a **TINYINT** data type, and insert the integer value of **0** for Salesforce’s boolean value of ‘**false**’, and insert the integer value of **1** for Salesforce’s boolean value of ‘**true**’.

General Differences between MySQL, Oracle, PostgreSQL and MS SQL Dialects

Insert Statements

Insert statements in MySQL, PostgreSQL and MS SQL

INSERT INTO tableName (col1,col2,col3) VALUES (‘val1’,’val2’,’val3’),(‘val4’,’val5’,’val6’) … ;

Insert statements in Oracle

INSERT ALL

INTO tableName(col1,col2,col3) VALUES (‘val1’,’val2’,’val3’)

INTO tableName(col1,col2,col3) VALUES (‘val4’,’val5’,’val6’)

SELECT 1 FROM DUAL;

Drop Table Statements

Drop table statement in MySQL and PostgreSQL

DROP TABLE IF EXISTS tableName;

Drop table statement in Oracle

BEGIN

EXECUTE IMMEDIATE 'DROP TABLE tableName' ;

EXCEPTION WHEN OTHERS THEN

IF SQLCODE != -942 THEN

RAISE;

END IF ;

END ;

Drop table statement in MS SQL

IF OBJECT_ID(tableName, ‘U’) IS NOT NULL

DROP TABLE tableName;

Create Table Statement

Defining an Auto-Incrementing Field
Define an auto-incrementing field in MySQL

In the field definition itself, one can concatenate the auto-increment expression: AUTO_INCREMENT

For example: ID int NOT NULL AUTO_INCREMENT

Define an auto-incrementing field in Oracle

In the field definition itself, one can concatenate the auto-increment expression: GENERATED ALWAYS AS IDENTITY

For example: "Revision" NUMBER NOT NULL GENERATED ALWAYS AS IDENTITY

Define an auto-incrementing field in PostgreSQL

In the field definition itself, instead of the column type, one can concatenate the auto-increment expression: SERIAL

For example: “ID” SERIAL NOT NULL

Define an auto-incrementing field in MS SQL

In the field definition itself, one can concatenate the auto-increment expression: IDENTITY(1,1)

For example: “ID” INT NOT NULL IDENTITY(1,1)

Defining a Primary Key
Defining a Primary Key in MySQL

After the field was defined in the creation statement one can define the field as a primary key in the following way:

PRIMARY KEY ("ID")

Defining a Primary Key in Oracle and in MS SQL

In the field definition itself, one can concatenate the primary key definition expression: PRIMARY KEY

For example: “ID” NUMBER PRIMARY KEY

Defining a Primary Key in PostgreSQL

After the field was defined in the creation statement one can define the field as a primary key in the following way:

PRIMARY KEY (“ID”)

Comments on Fields
Comments on Fields in MySQL

In the field definition itself, one can concatenate the comment part, as: COMMENT ‘comment string’

For example: id int(10) NOT NULL auto_increment COMMENT ‘unique ID for each foo entry’,

Comments on Fields in Oracle and in PostgreSQL

After the field was defined in the creation statement one can add a comment to the field in the following way:

COMMENT ON COLUMN “TableName”.”ColumnName” IS “Actual comment”;

Comments on Fields in MS SQL

After the field was defined in the creation statement one can add a comment to the field using a stored procedure in the following way:

```
EXEC sys.sp_addextendedproperty

@name = N'field_label',

@value = N'Actual comment',

@level0type = N'Schema',

@level0name = N'dbo',

@level2type = N'Column',

@level2name = N'ColumnName';
```

Index Fields
Index Fields in MySQL

After the field was defined in the creation statement one can add an index on the field in the following way:

```
KEY `Index_name` (`field_name`)
```

Index Fields in Oracle and in PostgreSQL

After the field was defined in the creation statement one can add an index on the field in the following way:

```
CREATE INDEX “index_name” ON “table_name”.”field_name”
```

Index Fields in MS SQL

After the field was defined in the creation statement one can add an index on the field in the following way:

```
CREATE INDEX “index_name” ON “table_name”(”field_name”)
```

Unique Fields
Unique Fields in MySQL, Oracle, and PostgreSQL

In the field definition itself, one can concatenate the ‘Unique’ definition expression: UNIQUE

For example: “ID” INT NOT NULL UNIQUE

Unique Fields in MS SQL

After the field has been defined in the creation statement, you can add a unique field constraint in the form of a "unique non-clustered" index, in the following way:

```
CREATE UNIQUE NONCLUSTERED INDEX “index_name” ON “table_name”(”field_name”)WHERE
“field_name” IS NOT NULL
```

Character Escaping
Character Escaping in MySQL and MS SQL

The following characters are escaped in the export using the ‘\’ character:

‘\’, “ ‘ “, ‘ “ ‘, ‘\r’, ‘\n’, ‘\t’.

Character Escaping in Oracle

In order to successfully escape characters in the Oracle export, an escape character definition is created before the table creation: SET ESCAPE '\\'

The following characters are escaped in the export using the '\' character:

'\', " ' ", ' " ', '&;', '_', '%'.

Character Escaping in PostgreSQL

In order to successfully escape characters in the PostgreSQL export, an escape character notation is prepended to every string type field value, in the following way:

E 'Inserted value that can now have escaped value such as \r'

The following characters are escaped in the export using the '\' character:

'\', " ' ", ' " ', '\r', '\n', '\t'.

Fields

Table 1. Fields

SalesForce	MySQL	Oracle	PostgreSQL	MS SQL
boolean	TINYINT(1)	NUMBER(1)	BOOLEAN	TINYINT
double	DOUBLE	FLOAT(24)	DOUBLE PRECISION	FLOAT(24)
dateTime	DATETIME	DATESTAMP	DATESTAMP	DATETIME
time	TIME	VARCHAR2(8)	TIME	TIME
date	DATE	DATE	DATE	DATE
address	VARCHAR(255)	VARCHAR(255)	VARCHAR(255)	VARCHAR(255)
ID	VARCHAR(#)	VARCHAR2(#)	VARCHAR(#)	VARCHAR(#)
int	INT(#)	NUMBER	INTEGER	INT
string	VARCHAR(#)	VARCHAR2(#)	VARCHAR(#)	VARCHAR(#)
picklist	VARCHAR(256)	VARCHAR(256)	VARCHAR(256)	VARCHAR(256)
anyType	VARCHAR(#)	VARCHAR2(#)	VARCHAR(#)	VARCHAR(#)
(Large string)	TEXT	CLOB	VARCHAR	VARCHAR(MAX)
(Max size string)	MEDIUMTEXT	CLOB	TEXT	VARCHAR(MAX)

Future Improvements

Oracle

Support for fields that are longer than 4k bytes. One possible workaround for a user is to use a single insert statement with the help of variables:

```
DECLARE

v_long_text CLOB;

BEGIN

v_long_text := 'your long string of text' ;

INSERT INTO table VALUES ( 1 , v_long_text);

END ;
```

http://stackoverflow.com/questions/8801814/how-to-insert-update-larger-size-of-data-in-the-oracle-tables