Netezza to BigQuery SQL translation reference

Contents

About this document

Highlights	
Purpose	To detail common similarities and differences in SQL syntax between Netezza and BigQuery, so as to help accelerate the planning and execution of moving a customer's enterprise data warehouse (EDW) to BigQuery.
Intended audience	Enterprise architects, DBAs, application developers, and IT security.
Key assumptions	That the audience is familiar with Netezza and is looking for guidance on transitioning to BigQuery.

Introduction

Netezza data warehousing is designed to work with Netezza-specific SQL syntax. Netezza SQL is based on Postgres 7.2. SQL scripts written for Netezza can't be used in a BigQuery data warehouse without alterations because the SQL dialects vary.

This document details the similarities and differences in SQL syntax between Netezza and BigQuery in the following areas:

- Data types
- SQL language elements
- Query syntax
- DML
- DDL
- Stored procedures
- Functions

Data types

Netezza	BigQuery	Notes
INTEGER/INT/INT4	INT64	
SMALLINT/INT2	INT64	
BYTEINT/INT1	INT64	
BIGINT/INT8	INT64	
DECIMAL	NUMERIC	The DECIMAL data type in Netezza is an alias for the NUMERIC data type.
NUMERIC	NUMERIC INT64	
NUMERIC(p,s)	NUMERIC	The NUMERIC type in BigQuery does not enforce custom digit or scale bounds (constraints) like Netezza does. BigQuery has fixed 9 digits after the decimal, while Netezza allows a custom setup. In Netezza, precision p can range from 1 to 38 and scale s from 0 to the precision.
FLOAT(p)	FLOAT64	
REAL/FLOAT(6)	FLOAT64	
DOUBLE PRECISION/FLOAT(14)	FLOAT64	
CHAR/CHARACTER	STRING	The STRING type in BigQuery is variable-length and does not require manually setting a max character length as the Netezza CHARACTER and VARCHAR types require. The default value of n in CHAR(n) is 1. The maximum character string size is 64,000.
VARCHAR	STRING	The STRING type in BigQuery is variable-length and does not require manually setting a max character length as the Netezza CHARACTER and VARCHAR types require. The maximum character string size is 64,000.
NCHAR	STRING	The STRING type in BigQuery is stored as variable-length UTF-8 encoded Unicode. The maximum length is 16,000 characters.
NVARCHAR	STRING	The STRING type in BigQuery is stored as variable-length UTF-8-encoded Unicode. The maximum length is 16,000 characters.
VARBINARY	BYTES	

ST_GEOMETRY	GEOGRAPHY	
BOOLEAN/BOOL	BOOL	The BOOL type in BigQuery can only accept TRUE/FALSE, unlike the BOOL type in Netezza, which can accept a variety of values like 0/1, yes/no, true/false, on/off.
DATE	DATE	
TIME	TIME	
TIMETZ/TIME WITH TIME ZONE	TIME	Netezza stores the TIME data type in UTC and allows you to pass an offset from UTC using the WITH TIME ZONE syntax. The TIME data type in BigQuery represents a time that's independent of any date or time zone.
TIMESTAMP	TIMESTAMP	The Netezza TIMESTAMP type has microsecond precision (including leap seconds) and is usually associated with the UTC time zone, the same as BigQuery. (Details)
	ARRAY	There is no ARRAY data type in Netezza. The ARRAY type is instead stored in a varchar field. (Details)

Netezza data formatting

For information about the default formats that Netezza SQL uses for each data type, see the Netezza data formatting documentation.

Timestamp and date type formatting

For more information about the date type formatting that Netezza SQL uses, see the Netezza date/time template patterns documentation. For more information about the date/time functions, see the Netezza date/time functions documentation.

When you convert date type formatting elements from Netezza to BigQuery standard SQL, you must pay particular attention to time zone differences between `TIMESTAMP` and `DATETIME`, as summarized in the following table.

Netezza		BigQuery
CURRENT_TIMESTAMP CURRENT_TIME	TIME information in Netezza can have different time zone information, which is defined using WITH TIME ZONE.	If possible, use CURRENT_TIMESTAMP, which is formatted correctly. However, the output format does show the UTC time zone (internally, BigQuery does not have a time zone). DATETIME in the bq command-line tool and Cloud Console is formatted using a T separator according to RFC 3339. However, in Python and Java JDBC, a space is used as a separator.

		Use the explicit FORMAT_DATETIME to define the date format correctly. Otherwise, an explicit cast is made to a string, for example: `CAST(CURRENT_DATETIME() AS STRING)` This also returns a space separator.
CURRENT_DATE	CURRENT_DATE	
CURRENT_DATE-3		BigQuery does not support arithmetic data operations. Instead, use DATE_ADD.

Query syntax

SELECT statement

For the most part, the Netezza `SELECT` statement is compatible with BigQuery. The following table contains a list of exceptions.

Netezza	BigQuery
SELECT without FROM clause	Supports special case such as: `SELECT 1 UNION ALL SELECT 2;`

`SELECT`
`  (subquery) AS flag,`
`  CASE WHEN flag = 1 THEN`
`...`
	In BigQuery, columns cannot reference the output of other columns defined within the same query. You must duplicate the logic or move the logic into a nested query.
	
	Option 1
	`SELECT`
	`  (subquery) AS flag,`
	`  CASE WHEN (subquery) = 1 THEN ...`
	
	Option 2
	`SELECT`
	`  q.*,`
	`  CASE WHEN flag = 1 THEN ...`
	`FROM (`
	`  SELECT`
	`  (subquery) AS flag,`
	`  ...`
	`) q`

Comparison operators

Netezza	BigQuery	Description
`exp = exp2`	`exp = exp2`	Equal
`exp <= exp2`	`exp <= exp2`	Less than or equal to
`exp < exp2`	`exp < exp2`	Less than
`exp <> exp2` `exp != exp2`	`exp <> exp2` `exp != exp2`	Not equal
`exp >= exp2`	`exp >= exp2`	Greater than or equal to
`exp > exp2`	`exp > exp2`	Greater than

Built-in SQL functions

Netezza	BigQuery	Description
`CURRENT_DATE`	`CURRENT_DATE`	Get the current date (year, month, and day).
`CURRENT_TIME`	`CURRENT_TIME`	Get the current time with fraction.
`CURRENT_TIMESTAMP`	`CURRENT_TIMESTAMP`	Get the current system date and time to the nearest full second.
`NOW`	`CURRENT_TIMESTAMP`	Get the current system date and time to the nearest full second.
`COALESCE(exp, 0)`	`COALESCE(exp, 0)`	Replace NULL with zero.
`NVL(exp, 0)`	`IFNULL(exp, 0)`	Replace NULL with zero.
`EXTRACT(DOY FROM timestamp_expression)`	`EXTRACT(DAYOFYEAR FROM timestamp_expression)`	Return the number of days from the beginning of the year.
`ADD_MONTHS(date_expr, num_expr)`	`DATE_ADD(date, INTERVAL k MONTH)`	Add months to a date.
`DURATION_ADD(date, k)`	`DATE_ADD(date, INTERVAL k DAY)`	Perform addition on dates.
`DURATION_SUBTRACT(date, k)`	`DATE_SUB(date, INTERVAL k DAY)`	Perform subtraction on dates.
`str1 || str2`	`CONCAT(str1, str2)`	Concatenate strings.

For more information, see the BigQuery function documentation.

Functions

Aggregate functions

Netezza	BigQuery
	ANY_VALUE
	APPROX_COUNT_DISTINCT
	APPROX_QUANTILES
	APPROX_TOP_COUNT
	APPROX_TOP_SUM
AVG	AVG
intNand	BIT_AND
intNnot	bitwise not operator: ~
intNor	BIT_OR
intNxor	BIT_XOR
intNshl	
intNshr	
CORR	CORR
COUNT	COUNT
	COUNTIF
COVAR_POP	COVAR_POP
COVAR_SAMP	COVAR_SAMP
GROUPING	
	LOGICAL_AND
	LOGICAL_OR
MAX	MAX
MIN	MIN
MEDIAN	PERCENTILE_CONT(x, 0.5)
STDDEV_POP	STDDEV_POP
STDDEV_SAMP	STDDEV_SAMP, STDDEV
	STRING_AGG
SUM	SUM
VAR_POP	VAR_POP
VAR_SAMP	VAR_SAMP, VARIANCE

Analytical functions

Netezza	BigQuery
	ANY_VALUE
	ARRAY_AGG
ARRAY_CONCAT	ARRAY_CONCAT_AGG
ARRAY_COMBINE	
ARRAY_COUNT	
ARRAY_SPLIT	
ARRAY_TYPE	
AVG	AVG
intNand	BIT_AND
intNnot	bitwise not operator: ~
intNor	BIT_OR
intNxor	BIT_XOR
intNshl	
intNshr	
CORR	CORR
COUNT	COUNT
	COUNTIF
COVAR_POP	COVAR_POP
COVAR_SAMP	COVAR_SAMP
CUME_DIST	CUME_DIST
DENSE_RANK	DENSE_RANK
FIRST_VALUE	FIRST_VALUE
LAG	LAG
LAST_VALUE	LAST_VALUE
LEAD	LEAD
AND	LOGICAL_AND
OR	LOGICAL_OR
MAX	MAX
MIN	MIN
	NTH_VALUE
NTILE	NTILE
PERCENT_RANK	PERCENT_RANK

PERCENTILE_CONT	PERCENTILE_CONT
PERCENTILE_DISC	PERCENTILE_DISC
RANK	RANK
ROW_NUMBER	ROW_NUMBER
STDDEV	STDDEV
STDDEV_POP	STDDEV_POP
STDDEV_SAMP	STDDEV_SAMP
	STRING_AGG
SUM	SUM
VARIANCE	VARIANCE
VAR_POP	VAR_POP
VAR_SAMP	VAR_SAMP / VARIANCE
WIDTH_BUCKET	

Date/time functions

Netezza	BigQuery
ADD_MONTHS	DATE_ADD / TIMESTAMP_ADD
AGE	
CURRENT_DATE	CURRENT_DATE CURRENT_DATETIME
CURRENT_TIME	CURRENT_TIME
CURRENT_TIME(p)	
CURRENT_TIMESTAMP	CURRENT_TIMESTAMP
CURRENT_TIMESTAMP(p)	DATE DATE_ADD DATE_DIFF DATE_FROM_UNIX_DATE DATE_SUB
DATE_TRUNC	DATE_TRUNC
DATE_PART	
	DATETIME
	DATETIME_ADD
	DATETIME_DIFF
	DATETIME_SUB
	DATETIME_TRUNC

DURATION_ADD	
DURATION_SUBTRACT	
EXTRACT	EXTRACT (DATE), EXTRACT (TIMESTAMP) FORMAT_DATE FORMAT_DATETIME FORMAT_TIME FORMAT_TIMESTAMP
LAST_DAY	DATE_SUB(DATE_TRUNC(DATE_ADD(date_expression, INTERVAL 1 MONTH), MONTH), INTERVAL 1 DAY)
MONTHS_BETWEEN	DATE_DIFF(date_expression, date_expression, MONTH)
NEXT_DAY	
NOW	
NUMTODSINTERVAL	
NUMTOYMINTERVAL	
OVERLAPS	PARSE_DATE PARSE_DATETIME PARSE_TIME PARSE_TIMESTAMP STRING TIME TIME_ADD TIME_DIFF TIME_SUB TIME_TRUNC
TIMEOFDAY	
TIMESTAMP	TIMESTAMP TIMESTAMP_ADD

	TIMESTAMP_DIFF
	TIMESTAMP_MICROS
	TIMESTAMP_MILLIS
	TIMESTAMP_SECONDS
	TIMESTAMP_SUB
	TIMESTAMP_TRUNC
TIMEZONE	
TO_DATE	PARSE_DATE
TO_TIMESTAMP	PARSE_TIMESTAMP
	UNIX_DATE
	UNIX_MICROS
	UNIX_MILLIS
	UNIX_SECONDS

String functions

Netezza	BigQuery
ASCII	TO_CODE_POINTS(string_expr)[OFFSET(0)]
	BYTE_LENGTH
	TO_HEX
	CHAR_LENGTH
	CHARACTER_LENGTH
	CODE_POINTS_TO_BYTES
BTRIM	
CHR	CODE_POINTS_TO_STRING([numeric_expr])
	CONCAT
DBL_MP	
DLE_DST	ENDS_WITH
	FORMAT
	FROM_BASE32
	FROM_BASE64
	FROM_HEX
HEX_TO_BINARY	
HEX_TO_GEOMETRY	
INITCAP	

INSTR	
INT_TO_STRING	
LE_DST	
LENGTH	LENGTH
LOWER	LOWER
LPAD	LPAD
LTRIM	LTRIM
	NORMALIZE
	NORMALIZE_AND_CASEFOLD
PRI_MP	REGEXP_CONTAINS
REGEXP_EXTRACT	REGEXP_EXTRACT
REGEXP_EXTRACT_ALL	REGEXP_EXTRACT_ALL
REGEXP_EXTRACT_ALL_SP	
REGEXP_EXTRACT_SP	
REGEXP_INSTR	STRPOS(col, REGEXP_EXTRACT())
REGEXP_LIKE	
REGEXP_MATCH_COUNT	
REGEXP_REPLACE	REGEXP_REPLACE
REGEXP_REPLACE_SP	IF(REGEXP_CONTAINS,1,0)
	REGEXP_EXTRACT
REPEAT	REPEAT
	REPLACE
REVERSE	REVERSE
RPAD	RPAD
RTRIM	RTRIM
	SAFE_CONVERT_BYTES_TO_STRING
SCORE_MP	
SEC_MP	
SOUNDEX	SPLIT
	STARTS_WITH
STRING_TO_INT	
STRPOS	STRPOS
SUBSTR	SUBSTR
	TO_BASE32
	TO_BASE64

TO_CHAR	
TO_DATE	
TO_NUMBER	
TO_TIMESTAMP	TO_CODE_POINTS
	TO_HEX
TRANSLATE	
TRIM	TRIM
UPPER	UPPER
UNICODE	
UNICODES	

Math functions

Netezza	BigQuery
ABS	ABS
ACOS	ACOS
	ACOSH
ASIN	ASIN
	ASINH
ATAN	ATAN
ATAN2	ATAN2
	ATANH
CEIL DCEIL	CEIL CEILING
COS	COS
	COSH
COT	
DEGREES	DIV
EXP	EXP
FLOOR DFLOOR	FLOOR
GREATEST	GREATEST
	IEEE_DIVIDE
	IS_INF
	IS_NAN

LEAST	LEAST
LN	LN
LOG	LOG
	LOG10
MOD	MOD
	NULLIF(expr, 0)
PI	ACOS(-1)
POW FPOW	POWER / POW
RADIANS	
RANDOM	RAND
ROUND	ROUND
	SAFE_DIVIDE
SETSEED	
SIGN	SIGN
SIN	SIN
	SINH
SQRT NUMERIC_SQRT	SQRT
TAN	TAN
	TANH
TRUNC	TRUNC
	IFNULL(expr, 0)

DML syntax

INSERT statement

Netezza	BigQuery
`INSERT INTO table VALUES (...);`	`INSERT INTO table (...) VALUES (...);` Netezza offers a DEFAULT keyword and other constraints for columns. In BigQuery, omitting column names in the INSERT statement is valid only if all columns are given.

`INSERT INTO table (...) VALUES (...); INSERT INTO table (...) VALUES (...);`	`INSERT INTO table VALUES (), ()` BigQuery imposes DML quotas, which restrict the number of DML statements you can execute daily. To make good use of your quota, consider the following approaches: • Combine multiple rows in a single `INSERT` statement, instead of one row per `INSERT`. • Combine multiple DML statements (including `INSERT`) using a `MERGE` statement. • Use `CREATE TABLE ... AS SELECT` to create and populate new tables.

DML scripts in BigQuery have slightly different consistency semantics than the equivalent statements in Netezza. Also note that BigQuery does not offer constraints and `DEFAULT` apart from `NOT NULL`.

For an overview of snapshot isolation and session and transaction handling, see the Consistency guarantees and transaction isolation section.

UPDATE statement

In Netezza, the `WHERE` clause is optional, but in BigQuery it is necessary.

Netezza	BigQuery
`UPDATE tbl SET tbl.col1=val1;`	Not supported without the `WHERE` clause. Use `WHERE true` to update all rows.
`UPDATE A SET y = B.y, z = B.z + 1 FROM B WHERE A.x = B.x AND A.y IS NULL;`	`UPDATE A SET y = B.y, z = B.z + 1 FROM B WHERE A.x = B.x AND A.y IS NULL;`
`UPDATE A alias SET x = x + 1 WHERE f(x) IN (0, 1)`	`UPDATE A SET x = x + 1 WHERE f(x) IN (0, 1)`
`UPDATE A SET z = B.z FROM B WHERE A.x = B.x AND A.y = B.y`	`UPDATE A SET z = B.z FROM B WHERE A.x = B.x AND A.y = B.y`

See `UPDATE` DML examples in BigQuery.

Because of DML quotas, you should prefer larger `MERGE` statements over multiple single `UPDATE` and `INSERT` statements. DML scripts in BigQuery have slightly different consistency semantics than equivalent statements in Netezza. For an overview on snapshot isolation and session and transaction handling, see the Consistency guarantees and transaction isolation section.

DELETE, TRUNCATE statements

The `DELETE` and `TRUNCATE` statements are both ways to remove rows from a table without affecting the table schema or indexes. The `TRUNCATE` command has the same effect as the `DELETE` command, but is much faster than the `DELETE` command for large tables. `TRUNCATE` is supported in Netezza but not supported in BigQuery. However, you can use `DELETE` statements in both Netezza and BigQuery.

In BigQuery, the `DELETE` statement must have a `WHERE` clause. In Netezza, the `WHERE` clause is optional. If the `WHERE` clause is not specified, all the rows in the Netezza table are deleted.

Netezza	BigQuery	Description

`BEGIN;`
`LOCK TABLE A IN EXCLUSIVE MODE;`
`DELETE FROM A;`
`INSERT INTO A SELECT * FROM B;`
`COMMIT;`
	Replacing the contents of a table with query output is the equivalent of a transaction. You can do this with either a query or a copy operation.
	
	`bq query --replace --destination_table tableA 'SELECT * FROM tableB WHERE ...'`
	`bq cp -f tableA tableB`
		Replace the contents of a table with the results of a query.

`DELETE FROM database.table`	`DELETE FROM table WHERE TRUE;`	In Netezza, when a `DELETE` statement is run, the rows are not deleted physically but only marked for deletion. Running `GROOM TABLE` or `nzreclaim` later on removes the rows marked for deletion and reclaims the corresponding disk space.
`GROOM TABLE`		Netezza uses `GROOM TABLE` command to reclaim disk space by removing rows marked for deletion.

MERGE statement

`MERGE` must match at most one source row for each target row. DML scripts in BigQuery have slightly different consistency semantics than the equivalent statements in Netezza. For an overview on snapshot isolation and session and transaction handling, see Consistency guarantees and transaction isolation section.

See `MERGE` DML Examples in BigQuery and `MERGE` DML Examples in Netezza.

DDL syntax

CREATE TABLE statement

Netezza	BigQuery	Description
`TEMP` `TEMPORARY`	With BigQuery's DDL support, you can create a table from the results of a query and specify its expiration at creation time. For example, for three days: `CREATE TABLE` `` `fh-bigquery.public_dump.vtemp` `` `OPTIONS(` `expiration_timestamp=TIMESTAMP_ADD(CURRENT_TIMESTAMP(), INTERVAL 3 DAY) )`	Create tables temporary to a session.
`ZONE MAPS`	Not supported.	Quick search for the `WHERE` condition.
`DISTRIBUTE ON`	`PARTITION BY`	Partitioning.
`ORGANIZE ON`	`CLUSTER BY`	BigQuery clustering is available only in partitioned tables. Both Netezza and BigQuery support up to four keys for clustering. Netezza clustered base tables (CBT) provide equal precedence to each of the clustering columns. BigQuery gives precedence to the first column on which the table is clustered, followed by the second column, and so on.
`ROW SECURITY`	Authorized View	Row-level security.
`CONSTRAINT`	Not supported.	Check constraints.

DROP statement

Netezza	BigQuery
DROP TABLE	DROP TABLE
DROP DATABASE	DROP DATABASE
DROP VIEW	DROP VIEW

Column options and attributes

Netezza	BigQuery	Description
NULL NOT NULL	NULLABLE REQUIRED	Specify if the column is allowed to contain NULL values.
REFERENCES	Not supported.	Specify column constraint.
UNIQUE	Not supported.	Each value in the column must be unique.
DEFAULT	Not supported.	Default value for all values in the column.

Temporary tables

Netezza supports TEMPORARY tables that exist for the duration of a session.

To build a temporary table in BigQuery, do the following:

1. Create a dataset that has a short time to live (for example, 12 hours).
2. Create the temporary table in the dataset, with a table name prefix of `temp`. For example, to create a table that expires in one hour, do this:

   ```
   CREATE TABLE temp.name (col1, col2, ...)
   OPTIONS(expiration_timestamp=TIMESTAMP_ADD(CURRENT_TIMESTAMP(),
   INTERVAL 1 HOUR));
   ```

   With scripting and procedures, the `expiration_timestamp` value can be a script-wide variable.
3. Start reading and writing from the temporary table.

More often, users prefer to remove duplicates independently in order to find errors in downstream systems.

Note that BigQuery does not support `DEFAULT` and `IDENTITY` (sequences) columns.

Procedural SQL statements

Netezza uses the NZPLSQL scripting language to work with stored procedures. NZPLSQL is based on Postgres PL/pgSQL language. This section describes how to convert procedural SQL statements used in stored procedures, functions, and triggers from Netezza to BigQuery.

CREATE PROCEDURE statement

Netezza supports stored procedures, but BigQuery does not.

Netezza	BigQuery	Description
`CREATE PROCEDURE`	Not supported.	
`<name-of-procedure>()`	Not supported.	Name of procedure. Parameters can be passed to the procedure inside the brackets.
`RETURNS datatype`	Not supported.	Returns either a unique value or a REFTABLE result set in the form of tables.
`LANGUAGE`	Not supported.	Language used by the stored procedure (NZPLSQL).
Multiple `BEGIN END` statements supported inside the `BEGIN_PROC END_PROC` block.	Not supported.	Number of returned result sets.
Declarations are inside `BEGIN END` block.	Not supported.	

Variable declaration and assignment

BigQuery scripting is a solution to replace procedural SQL statements. However, it does not provide an explicit transaction boundary or sessions, and with scripting, all commands have to be executed within a single network call.

Netezza	BigQuery	Description
`DECLARE var datatype(len) [DEFAULT value];`	Scripting and procedure.	Declare variable.
`SET var = value;`	Not supported.	Assign value to variable.

Exception handlers

Netezza supports exception handlers that can be triggered for certain error conditions. BigQuery does not support condition handlers.

Netezza	BigQuery	Description
EXCEPTION	Not supported.	Declare SQL exception handler for general errors.

Dynamic SQL statements

Netezza supports dynamic SQL queries inside stored procedures. The scripting feature in BigQuery supports dynamic SQL statements like those shown in the following table.

Netezza	BigQuery	Description
EXECUTE IMMEDIATE *sql_str*;	EXECUTE IMMEDIATE	Execute dynamic SQL.

Flow-of-control statements

BigQuery scripting enables you to use control flow statements such as `IF` and `WHILE`.

Netezza	BigQuery	Description

IF THEN ELSE STATEMENT
IF *condition* THEN
...
ELSE
...
END IF;
	IF *condition* THEN
	stmts ELSE *stmts*
	END IF
		Execute conditionally.

Iterative Control FOR *var* AS SELECT ... DO *stmts* END FOR; FOR *var* AS *cur* CURSOR FOR SELECT ... DO *stmts* END FOR;	Not supported.	Iterate over a collection of rows.
Iterative Control LOOP *stmts* END LOOP;	LOOP sql_statement_list END LOOP;	Loop block of statements.
EXIT WHEN	BREAK	Exit a procedure.
WHILE *condition* LOOP	WHILE *condition* DO *stmts* END WHILE	Execute a loop of statements until a while condition fails.

Other statements and procedural language elements

Netezza	BigQuery	Description
CALL *proc(param,...)*	Not supported.	Execute a procedure.
EXEC proc(param,...)	Not supported.	Execute a procedure.
EXECUTE proc(param,...)	Not supported.	Execute a procedure.

Multi-statement and multi-line SQL statements

Netezza supports transactions (sessions) and therefore supports statements separated by semicolons that are consistently executed together.

Netezza	BigQuery	Description
; // SEMICOLON	Supported in scripting and stored procedures.	Separate one statement from another.

Other SQL statements

Netezza	BigQuery	Description
GENERATE STATISTICS		Generate statistics for all the tables in the current database.
GENERATE STATISTICS ON *table_name*		Generate statistics for a specific table.
GENERATE STATISTICS ON *table_name(col1,col4)*	Either use statistical functions like MIN, MAX, and AVG, use the Cloud Console, or use Cloud Data Loss Prevention.	Generate statistics for specific columns in a table.
GENERATE STATISTICS ON *table_name*	APPROX_COUNT_DISTINCT(*col*)	Show the number of unique values for columns.
INSERT INTO *table_name*	INSERT INTO *table_name*	Insert a row.
LOCK TABLE *table_name* FOR EXCLUSIVE;	Not supported.	Lock a row.
SET SESSION CHARACTERISTICS AS TRANSACTION ISOLATION LEVEL ...	BigQuery uses snapshot isolation. For details, see Consistency guarantees in this document.	Define the transaction isolation level.
BEGIN TRANSACTION END TRANSACTION COMMIT	BigQuery uses snapshot isolation. For details, see Consistency guarantees in this document.	Define the transaction boundary for multi-statement requests.

`EXPLAIN ...`	Not supported. Similar features are the query plan explanation in the Cloud Console and the slot allocation and audit logging in Cloud Monitoring.	Show query plan for a SELECT statement.
`User_Views` metadata `System_Views` metadata	`SELECT * EXCEPT(is_typed) FROM mydataset.INFORMATION_SCHEMA.TABLES` BigQuery `INFORMATION_SCHEMA`.	Query objects in the database.

Consistency guarantees and transaction isolation

Both Netezza and BigQuery are atomic—that is, ACID compliant on a per-mutation level across many rows. For example, a `MERGE` operation is completely atomic, even with multiple inserted values. BigQuery has no concept of a session or of an explicit transaction boundary and therefore does not support multi-statement transactions or index consistency.

Transactions

Netezza syntactically accepts all four modes of ANSI SQL transaction isolation. But regardless of what mode is specified, only `SERIALIZABLE` is used to maximize consistency and avoid dirty, nonrepeatable, and phantom reads between concurrent transactions. Netezza does not use conventional locking to enforce consistency. Instead, it uses serialization dependency checking, a form of optimistic concurrency control, to automatically roll back the latest transaction when two transactions attempt to modify the same data.

BigQuery supports optimistic concurrency control (first to commit wins) with snapshot isolation (the query sees the last committed data before it started). This approach guarantees the same level of consistency on a per-row, per-mutation basis and across rows within the same DML statement. The concurrency limitations of DML in BigQuery essentially guarantee no conflicts between two updates. However, BigQuery does not provide an explicit transaction boundary or session.

Rollback

Netezza supports `ROLLBACK` to abort the current transaction and roll back all the changes made in the transaction. There is no concept of an explicit rollback in BigQuery. The workarounds are table decorators or using `FOR SYSTEM_TIME AS OF`.

Database limits

The following table shows Netezza database maximums.

Limit	Netezza	BigQuery
Tables per database	32,000	Unrestricted
Columns per table	1600	10,000
Maximum row size	64 KB	100 MB
Column and table name length	128 bytes	16,384 Unicode characters
Rows per table	Unlimited	Unlimited
Maximum SQL request length		1 MB (maximum unresolved standard SQL query length). 12 MB (maximum resolved legacy and standard SQL query length). Streaming: 10 MB (HTTP request size limit) 10,000 (maximum rows per request)
Maximum request and response size		10 MB (request) and 10 GB (response) or virtually unlimited if using pagination or the BigQuery Storage API.
Maximum number of concurrent sessions	63 concurrent read-write transactions. 2,000 concurrent connections to the server.	100 concurrent queries (can be raised with slot reservation). 300 concurrent API requests per user.