Oracle® to BigQuery SQL translation reference

About this document

This document details the similarities and differences in SQL syntax between Oracle and BigQuery to help you accelerate the planning and execution of moving your enterprise data warehouse (EDW) to BigQuery in Google Cloud. Scripts written for Oracle might need to be altered before you can use them in BigQuery because the SQL dialects vary between the services. Google Cloud partners have tools for automating the conversion of Oracle SQL scripts. If you are interested in learning more, contact your account representative.

Note: In some cases, there is no direct mapping between a SQL element in Oracle and BigQuery. However, in most cases, you can achieve the same functionality in BigQuery that you can in Oracle using an alternative means, as shown in the examples in this document.

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

Data types

This section shows equivalents between data types in Oracle and in BigQuery.

Note: Oracle supports `DEFAULT` and other constraints; these are not used in BigQuery.

Oracle data type	BigQuery data type mapping	BigQuery type conversion description
`VARCHAR2`	`STRING`	
`NVARCHAR2`		

CHAR NCHAR CLOB NCLOB		
INTEGER SHORTINTEGER LONGINTEGER	INT64	
NUMBER NUMBER(*, x)	NUMERIC	BigQuery does not allow user specification of custom values for precision or scale. As a result, a column in Oracle might be defined so that it has a bigger scale than BigQuery supports. Additionally, before storing a decimal number, Oracle rounds up if that number has more digits after the decimal point than are specified for the corresponding column. In BigQuery, you can implement this feature by using the ROUND() function.
NUMBER(x, -y) NUMBER(x)	INT64	If a user tries to store a decimal number, Oracle rounds it up to a whole number. For BigQuery, an attempt to store a decimal number in a column defined as INT64 results in an error. In this case, apply the ROUND() function. BigQuery INT64 data types allow up to 18 digits of precision. If a number field has more than 18 digits, use the FLOAT64 data type in BigQuery.
FLOAT BINARY_DOUBLE BINARY_FLOAT	FLOAT64/NUMERIC	FLOAT is an exact data type, and it's a NUMBER subtype in Oracle. In BitQuery, FLOAT64 is an approximate data type. NUMERIC is often a better match for FLOAT type in BigQuery.
LONG	BYTES	The LONG data type is used in earlier versions and is not suggested in new versions of Oracle Database. The BYTES data type in BigQuery can be used if it is necessary to hold LONG data in BigQuery. A better approach is putting binary objects in Cloud Storage and holding references in BigQuery.
BLOB	BYTES	The BYTES data type can be used to store variable-length binary data. If this field is not queried and not used in analytics, a better option is to store binary data in Cloud

		Storage.
BFILE	[STRING]()	Binary files can be stored in Cloud Storage, and the STRING data type can be used for referencing files in a BigQuery table.
DATE	[DATE]()	
TIMESTAMP TIMESTAMP(x) TIMESTAMP WITH TIME ZONE TIMESTAMP WITH LOCAL TIME ZONE	[TIMESTAMP]()	BigQuery supports microsecond precision (10^{-6}), in comparison to Oracle, which supports precision ranging from 0 to 9. BigQuery supports a time zone region name from a TZ database and time zone offset from UTC. In BigQuery, use a manual time zone conversion to match Oracle's TIMESTAMP WITH LOCAL TIME ZONE feature.
INTERVAL YEAR TO MONTH INTERVAL DAY TO SECOND	[STRING]()	Interval values can be stored as a STRING data type in BigQuery.
RAW LONG RAW	[BYTES]()	The BYTES data type can be used to store variable-length binary data. If this field is not queried and used in analytics, a better option is to store binary data on Cloud Storage.
ROWID	[STRING]()	These data types are used by Oracle internally to specify unique addresses to rows in a table. Normally ROWID or UROWID fields should not be used in applications. But if this is the case, the STRING data type can be used to hold this data.

Type formatting

Oracle SQL uses a set of default formats that are set as parameters for displaying expressions and column data, and for conversions between data types. For example, NLS_DATE_FORMAT set as YYYY/MM/DD formats dates as YYYY/MM/DD by default. You can find more information about the NLS settings in the Oracle online documentation. In BigQuery, there are no initialization parameters.

By default, BigQuery expects all source data to be UTF-8 encoded when loading. Optionally, if you have CSV files with data encoded in ISO-8859-1 format, you can explicitly specify the encoding when you import your data so that BigQuery can properly convert your data to UTF-8 during the import process.

It is only possible to import data that is ISO-8859-1 or UTF-8 encoded. BigQuery stores and returns the data as UTF-8 encoded. Intended date format or timezone can be set in DATE and TIMESTAMP functions.

Timestamp and date type formatting

When you convert timestamp and date formatting elements from Oracle to BigQuery, you must pay attention to timezone differences between TIMESTAMP and DATETIME as summarized in the following table.

Notice there are no parentheses in the Oracle formats because the formats (CURRENT_*) are keywords, not functions.

Oracle		BigQuery
CURRENT_TIMESTAMP	TIMESTAMP information in Oracle can have different time zone information, which is defined by using WITH TIME ZONE in the column definition or by setting the TIME_ZONE variable.	If possible, use CURRENT_TIMESTAMP(), which is formatted in ISO format. However, the output format does show the UTC timezone. (Internally, BigQuery does not have a timezone.) Note the following details on differences in the ISO format: DATETIME is formatted based on output channel conventions. In the bq command-line tool and Cloud Console, DATETIME is formatted using a T separator according to RFC 3339. However, in Python and Java JDBC, a space is used as a separator. If you want to use an explicit format, use FORMAT_DATETIME(), which makes an explicit cast a string. For example, the following expression returns a space separator: CAST(CURRENT_DATETIME() AS STRING) Oracle supports a DEFAULT keyword in TIME columns to set the current time (timestamp); this is not used in BigQuery.
CURRENT_DATE SYSDATE	Oracle uses two types for date: type 12 and type 13. Oracle uses type 12 when storing dates; internally, these are numbers with fixed length. Oracle uses type 13 when a date is returned by SYSDATE or CURRENT_DATE.	BigQuery has a separate DATE format that returns a date in ISO 8601 format. DATE_FROM_UNIX_DATE can't be used because it is 1970-based. Oracle supports a DEFAULT keyword in DATE columns to set the current date; this is not used in BigQuery.
CURRENT_DATE-3	Date values are represented as integers. Oracle supports	For date types, use DATE_ADD() or DATE_SUB().

	arithmetic operators for date types.	BigQuery uses arithmetic operators for data types: INT64, NUMERIC, and FLOAT64.
NLS_DATE_FORMAT	Set the session or system date format.	BigQuery uses ISO 8601, so make sure that you convert Oracle dates and times.

Query syntax

This section addresses differences in query syntax between Oracle and BigQuery.

SELECT statement

Most Oracle SELECT statements are compatible with BigQuery.

Functions, operators, and expressions

The following sections list mappings between Oracle functions and BigQuery equivalents.

Comparison operators

Oracle and BigQuery comparison operators are ANSI SQL:2011 compliant. The comparison operators in the following table are the same in both BigQuery and Oracle. You can use REGEXP_CONTAINS instead of REGEXP_LIKE in BigQuery.

Operator	Description
"="	Equal
<>	Not equal
!=	Not equal
>	Greater than
>=	Greater than or equal
<	Less than
<=	Less than or equal
IN ( )	Matches a value in a list
NOT	Negates a condition
BETWEEN	Within a range (inclusive)
IS NULL	NULL value
IS NOT NULL	Non-NULL value
LIKE	Pattern matching with %
EXISTS	Condition is met if subquery returns at least one row

The operators in the table are the same both in BigQuery and Oracle.

Logical expressions and functions

Oracle	BigQuery
CASE	CASE
COALESCE	COALESCE(expr1, ..., exprN)
DECODE	CASE.. WHEN.. END
NANVL	IFNULL
FETCH NEXT	LIMIT
NULLIF	NULLIF(expression, expression_to_match)
NVL	IFNULL(expr, 0), COALESCE(exp, 0)
NVL2	IF(cond, true_result, else_result)

Aggregate functions

The following table shows mappings between common Oracle aggregate, statistical aggregate, and approximate aggregate functions with their BigQuery equivalents.

Oracle	BigQuery
APPROX_COUNT	HLL_COUNT: set of functions with specified precision.
APPROX_COUNT_DISTINCT	APPROX_COUNT_DISTINCT
APPROX_COUNT_DISTINCT_AGG	APPROX_COUNT_DISTINCT
APPROX_COUNT_DISTINCT_DETAIL	APPROX_COUNT_DISTINCT
APPROX_PERCENTILE(percentile) WITHIN GROUP (ORDER BY expression)	APPROX_QUANTILES(expression, 100)[OFFSET(CAST(TRUNC(percentile * 100) as INT64))] BigQuery doesn't support the rest of the arguments that Oracle defines.
APPROX_PERCENTILE_AGG	APPROX_QUANTILES(expression, 100)[OFFSET(CAST(TRUNC(percentile * 100) as INT64))]
APPROX_PERCENTILE_DETAIL	APPROX_QUANTILES(expression, 100)[OFFSET(CAST(TRUNC(percentile * 100) as INT64))]
APPROX_SUM	APPROX_TOP_SUM(expression, weight, number)
AVG	AVG
BIT_COMPLEMENT	bitwise not operator: ~
BIT_OR	BIT_OR, X | Y

BIT_XOR	BIT_XOR, X ^ Y
BITAND	BIT_AND, X & Y
CARDINALITY	COUNT
COLLECT	BigQuery doesn't support TYPE AS TABLE OF. Consider using STRING_AGG() or ARRAY_AGG() in BigQuery.
CORR/CORR_K/ CORR_S	CORR
COUNT	COUNT
COVAR_POP	COVAR_POP
COVAR_SAMP	COVAR_SAMP
FIRST	Does not exist implicitly in BigQuery. Consider using UDFs.
GROUP_ID	Not used in BigQuery.
GROUPING	Not used in BigQuery.
GROUPING_ID	Not used in BigQuery.
LAST	Does not exist implicitly in BigQuery. Consider using UDFs.
LISTAGG	STRING_AGG, ARRAY_CONCAT_AGG(expression [ORDER BY key [{ASC|DESC}] [, ...]] [LIMIT n])
MAX	MAX
MIN	MIN
OLAP_CONDITION OLAP_EXPRESSION OLAP_EXPRESSION_BOOL OLAP_EXPRESSION_DATE OLAP_EXPRESSION_TEXT OLAP_TABLE POWERMULTISET POWERMULTISET_BY_CARDINALITY QUALIFY	Oracle specific and does not exist in BigQuery.
REGR_AVGX	AVG(IF(dep_var_expr is NULL OR ind_var_expr is NULL, NULL, ind_var_expr))
REGR_AVGY	AVG(IF(dep_var_expr is NULL OR ind_var_expr is NULL, NULL, dep_var_expr))

REGR_COUNT	SUM(IF(dep_var_expr is NULL OR ind_var_expr is NULL, NULL, 1))
REGR_INTERCEPT	AVG(dep_var_expr) - AVG(ind_var_expr) * (COVAR_SAMP(ind_var_expr,dep_var_expr) / VARIANCE(ind_var_expr))
REGR_R2	(COUNT(dep_var_expr) * SUM(ind_var_expr * dep_var_expr) - SUM(dep_var_expr) * SUM(ind_var_expr)) / SQRT((COUNT(ind_var_expr) * SUM(POWER(ind_var_expr, 2)) * POWER(SUM(ind_var_expr),2)) * (COUNT(dep_var_expr) * SUM(POWER(dep_var_expr, 2)) * POWER(SUM(dep_var_expr), 2)))
REGR_SLOPE(dep_var_expr, ind_var_expr)	COVAR_SAMP(ind_var_expr, dep_var_expr) / VARIANCE(ind_var_expr)
REGR_SXX	SUM(POWER(ind_var_expr, 2)) - COUNT(ind_var_expr) * POWER(AVG(ind_var_expr),2)
REGR_SXY	SUM(ind_var_expr*dep_var_expr) - COUNT(ind_var_expr) * AVG(ind) * AVG(dep_var_expr)
REGR_SYY	SUM(POWER(dep_var_expr, 2)) - COUNT(dep_var_expr) * POWER(AVG(dep_var_expr),2)
ROLLUP	ROLLUP
STDDEV_POP	STDDEV_POP
STDDEV_SAMP	STDDEV_SAMP, STDDEV
SUM	SUM
VAR_POP	VAR_POP
VAR_SAMP	VAR_SAMP, VARIANCE
WM_CONCAT	STRING_AGG

BigQuery offers the following additional aggregate functions:

- ANY_VALUE
- APPROX_TOP_COUNT
- COUNTIF
- LOGICAL_AND

- LOGICAL_OR

Analytical functions

The following table shows mappings between common Oracle analytic and aggregate analytic functions with their BigQuery equivalents.

Oracle	BigQuery
AVG	AVG
BIT_COMPLEMENT	bitwise not operator: ~
BIT_OR	BIT_OR / X | Y
BIT_XOR	BIT_XOR / X ^ Y
BITAND	BIT_AND / X & Y
BOOL_TO_INT	CAST(X AS INT64)
COUNT	COUNT
COVAR_POP	COVAR_POP
COVAR_SAMP	COVAR_SAMP
CUBE_TABLE	Not supported in BigQuery. Consider using a BI tool or a custom UDF.
CUME_DIST	CUME_DIST
DENSE_RANK (ANSI)	DENSE_RANK
FEATURE_COMPARE FEATURE_DETAILS FEATURE_ID FEATURE_SET FEATURE_VALUE	Does not exist implicitly in BigQuery. Consider using UDFs and BigQuery ML.
FIRST_VALUE	FIRST_VALUE
HIER_CAPTION HIER_CHILD_COUNT HIER_COLUMN HIER_DEPTH HIER_DESCRIPTION HIER_HAS_CHILDREN HIER_LEVEL HIER_MEMBER_NAME HIER_ORDER HIER_UNIQUE_MEMBER_NAME	Hierarchical queries are not supported in BigQuery.

LAST_VALUE	LAST_VALUE
LAG	LAG
LEAD	LEAD
LISTAGG	ARRAY_AGG STRING_AGG ARRAY_CONCAT_AGG
MATCH_NUMBER	Pattern recognition and calculation can be done with regular expressions and UDFs in BigQuery.
MATCH_RECOGNIZE	Pattern recognition and calculation can be done with regular expressions and UDFs in BigQuery.
MAX	MAX
MEDIAN	PERCENTILE_CONT(x, 0.5 RESPECT NULLS) OVER()
MIN	MIN
NTH_VALUE	NTH_VALUE (value_expression, constant_integer_expression [{RESPECT | IGNORE} NULLS])
NTILE	NTILE(constant_integer_expression)
PERCENT_RANK PERCENT_RANKM	PERCENT_RANK
PERCENTILE_CONT PERCENTILE_DISC	PERCENTILE_CONT
PERCENTILE_CONT PERCENTILE_DISC	PERCENTILE_DISC
PRESENTNNV PRESENTV PREVIOUS	Oracle specific and does not exist in BigQuery.
RANK (ANSI)	RANK
RATIO_TO_REPORT(expr) OVER (partition clause)	expr / SUM(expr) OVER (partition clause)
ROW_NUMBER	ROW_NUMBER
STDDEV_POP	STDDEV_POP
STDDEV_SAMP	STDDEV_SAMP, STDDEV
SUM	SUM
VAR_POP	VAR_POP
VAR_SAMP	VAR_SAMP, VARIANCE
VARIANCE	VARIANCE()
WIDTH_BUCKET	UDF can be used.

Date/time functions

The following table shows mappings between common Oracle date/time functions and their BigQuery equivalents.

Oracle	BigQuery

`ADD_MONTHS(date, integer)`	`DATE_ADD(date, INTERVAL integer MONTH),`
	If date is a `TIMESTAMP`, you can use
	`EXTRACT(DATE FROM TIMESTAMP_ADD(date, INTERVAL integer MONTH)).`

`CURRENT_DATE`	`CURRENT_DATE`
`CURRENT_TIME`	`CURRENT_TIME`
`CURRENT_TIMESTAMP`	`CURRENT_TIMESTAMP`
`DATE - k`	`DATE_SUB(date_expression, INTERVAL k DAY)`
`DATE + k`	`DATE_ADD(date_expression, INTERVAL k DAY)`
`DBTIMEZONE`	BigQuery does not support the database time zone.
`EXTRACT`	`EXTRACT(DATE)`, `EXTRACT(TIMESTAMP)`

`LAST_DAY`	`DATE_SUB(`
	`DATE_TRUNC(`
	`DATE_ADD(`
	`date_expression,`
	`INTERVAL 1 MONTH`
	`),`
	`MONTH`
	`),`
	`INTERVAL 1 DAY`
	`)`

[`LOCALTIMESTAMP`]()	BigQuery doesn't support time zone settings.
`MONTHS_BETWEEN`	`DATE_DIFF(date_expression, date_expression, MONTH)`

[`NEW_TIME`]()	`DATE(timestamp_expression, timezone)`
	`TIME(timestamp, timezone)`
	`DATETIME(timestamp_expression, timezone)`

`NEXT_DAY`	`DATE_ADD(`
	  `DATE_TRUNC(`
	    `date_expression,`
	    `WEEK(day_value)`
	  `),`
	  `INTERVAL 1 WEEK`
	`)`

`SYS_AT_TIME_ZONE`	`CURRENT_DATE([time_zone])`
`SYSDATE`	`CURRENT_DATE()`

SYSTIMESTAMP	CURRENT_TIMESTAMP()
TO_DATE	PARSE_DATE
TO_TIMESTAMP	PARSE_TIMESTAMP
TO_TIMESTAMP_TZ	PARSE_TIMESTAMP
TZ_OFFSET	Not supported in BigQuery. Consider using a custom UDF.
WM_CONTAINS WM_EQUALS WM_GREATERTHAN WM_INTERSECTION WM_LDIFF WM_LESSTHAN WM_MEETS WM_OVERLAPS WM_RDIFF	Periods are not used in BigQuery. UDFs can be used to compare two periods.

BigQuery offers the following additional date/time functions:

- CURRENT_DATETIME
- DATE_FROM_UNIX_DATE
- DATE_TRUNC
- DATETIME
- DATETIME_ADD
- DATETIME_DIFF
- DATETIME_SUB
- DATETIME_TRUNC
- FORMAT_DATE
- FORMAT_DATETIME
- FORMAT_TIME
- FORMAT_TIMESTAMP
- PARSE_DATETIME
- PARSE_TIME
- STRING
- TIME
- TIME_ADD
- TIME_DIFF
- TIME_SUB
- TIME_TRUNC
- TIMESTAMP
- TIMESTAMP_ADD
- TIMESTAMP_DIFF
- TIMESTAMP_MICROS
- TIMESTAMP_MILLIS
- TIMESTAMP_SECONDS
- TIMESTAMP_SUB
- TIMESTAMP_TRUNC
- UNIX_DATE
- UNIX_MICROS
- UNIX_MILLIS
- UNIX_SECONDS

String functions

The following table shows mappings between Oracle string functions and their BigQuery equivalents.

Oracle	BigQuery
ASCII	TO_CODE_POINTS(string_expr)[OFFSET(0)]
ASCIISTR	BigQuery doesn't support UTF-16.
RAWTOHEX	TO_HEX

LENGTH	CHAR_LENGTH
LENGTH	CHARACTER_LENGTH
CHR	CODE_POINTS_TO_STRING([mod(numeric_expr, 256)])
COLLATION	Doesn't exist in BigQuery. BigQuery doesn't support COLLATE in DML.
COMPOSE	Custom UDF.
CONCAT, (|| operator)	CONCAT
DECOMPOSE	Custom UDF.
ESCAPE_REFERENCE (UTL_I18N)	Is not supported in BigQuery. Consider using a UDF.
INITCAP	Custom UDF.
INSTR/INSTR2/INSTR4/INSTRB/INSTRC	Custom UDF.
LENGTH/LENGTH2/LENGTH4/LENGTHB/LENGTHC	LENGTH
LOWER	LOWER
LPAD	LPAD
LTRIM	LTRIM
NLS_INITCAP	Custom UDF.
NLS_LOWER	LOWER
NLS_UPPER	UPPER
NLSSORT	Oracle specific and does not exist in BigQuery.
POSITION	STRPOS(string, substring)
PRINTBLOBTOCLOB	Oracle specific and does not exist in BigQuery.
REGEXP_COUNT	ARRAY_LENGTH(REGEXP_EXTRACT_ALL(value, regex))
REGEXP_INSTR	STRPOS(source_string, REGEXP_EXTRACT(source_string, regexp_string)) **Note:** Returns first occurrence.
REGEXP_REPLACE	REGEXP_REPLACE
REGEXP_LIKE	IF(REGEXP_CONTAINS,1,0)
REGEXP_SUBSTR	REGEXP_EXTRACT, REGEXP_EXTRACT_ALL
REPLACE	REPLACE
REVERSE	REVERSE
RIGHT	SUBSTR(source_string, -1, length)

RPAD	RPAD
RTRIM	RTRIM
SOUNDEX	Not supported in BigQuery. Consider using a custom UDF.
STRTOK	SPLIT(instring, delimiter)[ORDINAL(tokennum)] **Note:** The entire delimiter string argument is used as a single delimiter. The default delimiter is a comma.
SUBSTR/SUBSTRB/SUBSTRC/SUBSTR2/SUBSTR4	SUBSTR
TRANSLATE	REPLACE
TRANSLATE USING	REPLACE
TRIM	TRIM
UNISTR	CODE_POINTS_TO_STRING
UPPER	UPPER
VERTICAL BARS, ||	CONCAT

BigQuery offers the following additional string functions:

- BYTE_LENGTH
- CODE_POINTS_TO_BYTES
- ENDS_WITH
- FROM_BASE32
- FROM_BASE64
- FROM_HEX
- NORMALIZE
- NORMALIZE_AND_CASEFOLD
- REPEAT
- SAFE_CONVERT_BYTES_TO_STRING
- SPLIT
- STARTS_WITH
- STRPOS
- TO_BASE32
- TO_BASE64
- TO_CODE_POINTS

Math functions

The following table shows mappings between Oracle math functions and their BigQuery equivalents.

Oracle	BigQuery
ABS	ABS
ACOS	ACOS
ACOSH	ACOSH
ASIN	ASIN
ASINH	ASINH
ATAN	ATAN

ATAN2	ATAN2
ATANH	ATANH
CEIL	CEIL
CEILING	CEILING
COS	COS
COSH	COSH
EXP	EXP
FLOOR	FLOOR
GREATEST	GREATEST
LEAST	LEAST
LN	LN
LNNVL	Use with ISNULL.
LOG	LOG
MOD (% operator)	MOD
POWER (** operator)	POWER, POW
DBMS_RANDOM.VALUE	RAND
RANDOMBYTES	Not supported in BigQuery. Consider using a custom UDF and RAND function.
RANDOMINTEGER	CAST(FLOOR(10*RAND()) AS INT64)
RANDOMNUMBER	Not supported in BigQuery. Consider using a custom UDF and RAND function.
REMAINDER	MOD
ROUND	ROUND
ROUND_TIES_TO_EVEN	ROUND()
SIGN	SIGN
SIN	SIN
SINH	SINH
SQRT	SQRT
STANDARD_HASH	FARM_FINGERPRINT, MD5, SHA1, SHA256, SHA512
STDDEV	STDDEV
TAN	TAN
TANH	TANH
TRUNC	TRUNC
NVL	IFNULL(expr, 0), COALESCE(exp, 0)

BigQuery offers the following additional math functions:

- DIV
- IEEE_DIVIDE
- IS_INF
- IS_NAN
- LOG10
- SAFE_DIVIDE

Type conversion functions

The following table shows mappings between Oracle type conversion functions and their BigQuery equivalents.

Oracle	BigQuery
BIN_TO_NUM	SAFE_CONVERT_BYTES_TO_STRING(value) CAST(x AS INT64)
BINARY2VARCHAR	SAFE_CONVERT_BYTES_TO_STRING(value)
CAST CAST_FROM_BINARY_DOUBLE CAST_FROM_BINARY_FLOAT CAST_FROM_BINARY_INTEGER CAST_FROM_NUMBER CAST_TO_BINARY_DOUBLE CAST_TO_BINARY_FLOAT CAST_TO_BINARY_INTEGER CAST_TO_NUMBER CAST_TO_NVARCHAR2 CAST_TO_RAW CAST_TO_VARCHAR	CAST(expr AS typename)
CHARTOROWID	Oracle specific, not needed.
CONVERT	BigQuery doesn't support character sets. Consider using a custom UDF.
EMPTY_BLOB	BLOB is not used in BigQuery.
EMPTY_CLOB	CLOB is not used in BigQuery.
FROM_TZ	Types with time zones are not supported in BigQuery. Consider using a UDF and FORMAT_TIMESTAMP.
INT_TO_BOOL	CAST
IS_BIT_SET	Does not exist implicitly in BigQuery. Consider using UDFs.
NCHR	UDF can be used to get the character equivalent in binary.
NUMTODSINTERVAL	INTERVAL data type is not supported in BigQuery.
NUMTOHEX	Not supported in BigQuery. Consider using a custom UDF and the TO_HEX function.
NUMTOHEX2	Not supported in BigQuery. Consider using a custom UDF and the TO_HEX function.

NUMTOYMINTERVAL	INTERVAL data type is not supported in BigQuery.
RAW_TO_CHAR	Oracle specific and does not exist in BigQuery.
RAW_TO_NCHAR	Oracle specific and does not exist in BigQuery.
RAW_TO_VARCHAR2	Oracle specific and does not exist in BigQuery.
RAWTOHEX	Oracle specific and does not exist in BigQuery.
RAWTONHEX	Oracle specific and does not exist in BigQuery.
RAWTONUM	Oracle specific and does not exist in BigQuery.
RAWTONUM2	Oracle specific and does not exist in BigQuery.
RAWTOREF	Oracle specific and does not exist in BigQuery.
REFTOHEX	Oracle specific and does not exist in BigQuery.
REFTORAW	Oracle specific and does not exist in BigQuery.
ROWIDTOCHAR	ROWID is an Oracle-specific type and does not exist in BigQuery. This value should be represented as a string.
ROWIDTONCHAR	ROWID is an Oracle-specific type and does not exist in BigQuery. This value should be represented as a string.
SCN_TO_TIMESTAMP	SCN is an Oracle-specific type and does not exist in BigQuery. This value should be represented as a timestamp.
TO_ACLID TO_ANYLOB TO_APPROX_COUNT_DISTINCT TO_APPROX_PERCENTILE TO_BINARYDOUBLE TO_BINARYFLOAT TO_BLOB TO_CHAR TO_CLOB TO_DATE TO_DSINTERVAL TO_LOB TO_MULTI_BYTE TO_NCHAR TO_NCLOB TO_NUMBER TO_RAW TO_SINGLE_BYTE TO_TIME TO_TIMESTAMP TO_TIMESTAMP_TZ TO_TIME_TZ TO_UTC_TIMEZONE_TZ TO_YMINTERVAL	CAST(expr AS typename) PARSE_DATE PARSE_TIMESTAMP Cast syntax is used in a query to indicate that the result type of an expression should be converted to some other type.

TREAT	Oracle specific and does not exist in BigQuery.
VALIDATE_CONVERSION	Not supported in BigQuery. Consider using a custom UDF.
VSIZE	Not supported in BigQuery. Consider using a custom UDF.

JSON functions

The following table shows mappings between Oracle JSON functions and their BigQuery equivalents.

Oracle	BigQuery
AS_JSON	TO_JSON_STRING(value[, pretty_print])
JSON_ARRAY	Consider using UDFs and TO_JSON_STRING function.
JSON_ARRAYAGG	Consider using UDFs and TO_JSON_STRING function.
JSON_DATAGUIDE	Custom UDF.
JSON_EQUAL	Custom UDF.
JSON_EXIST	Consider using UDFs and JSON_EXTRACT or JSON_EXTRACT_SCALAR.
JSON_MERGEPATCH	Custom UDF.
JSON_OBJECT	Not supported in BigQuery.
JSON_OBJECTAGG	Not supported in BigQuery.
JSON_QUERY	Consider using UDFs and JSON_EXTRACT or JSON_EXTRACT_SCALAR.
JSON_TABLE	Custom UDF.
JSON_TEXTCONTAINS	Consider using UDFs and JSON_EXTRACT or JSON_EXTRACT_SCALAR.
JSON_VALUE	JSON_EXTRACT_SCALAR

XML functions

BigQuery does not provide implicit XML functions. XML can be loaded to BigQuery as a string, and UDFs can be used to parse XML. Alternatively, XML processing can be done by an ETL/ELT tool. The following table shows Oracle XML functions.

Oracle	BigQuery
DELETEXML	BigQuery UDFs or an ETL tool like Dataflow can be used to process XML.
ENCODE_SQL_XML	
EXISTSNODE	
EXTRACTCLOBXML	
EXTRACTVALUE	
INSERTCHILDXML	

INSERTCHILDXMLAFTER	
INSERTCHILDXMLBEFORE	
INSERTXMLAFTER	
INSERTXMLBEFORE	
SYS_XMLAGG	
SYS_XMLANALYZE	
SYS_XMLCONTAINS	
SYS_XMLCONV	
SYS_XMLEXNSURI	
SYS_XMLGEN	
SYS_XMLI_LOC_ISNODE	
SYS_XMLI_LOC_ISTEXT	
SYS_XMLINSTR	
SYS_XMLLOCATOR_GETSVAL	
SYS_XMLNODEID	
SYS_XMLNODEID_GETLOCATOR	
SYS_XMLNODEID_GETOKEY	
SYS_XMLNODEID_GETPATHID	
SYS_XMLNODEID_GETPTRID	
SYS_XMLNODEID_GETRID	
SYS_XMLNODEID_GETSVAL	
SYS_XMLT_2_SC	
SYS_XMLTRANSLATE	
SYS_XMLTYPE2SQL	
UPDATEXML	
XML2OBJECT	
XML2OBJECT	
XMLCAST	
XMLCDATA	
XMLCOLLATVAL	
XMLCOMMENT	
XMLCONCAT	
XMLDIFF	
XMLELEMENT	
XMLEXISTS	

XMLEXISTS2	
XMLFOREST	
XMLISNODE	
XMLISVALID	
XMLPARSE	
XMLPATCH	
XMLPI	
XMLQUERY	
XMLQUERYVAL	
XMLSERIALIZE	
XMLTABLE	
XMLTOJSON	
XMLTRANSFORM	
XMLTRANSFORMBLOB	
XMLTYPE	

ML functions

ML functions in Oracle and BigQuery are different. Oracle requires an Advanced Analytics pack and licenses to do ML on the database. Oracle uses the `DBMS_DATA_MINING` package for ML. Converting Oracle Data Mining jobs might require rewriting the code. You can choose from comprehensive Google AI product offerings such as BigQuery ML, AI APIs (including Speech-to-Text, Text-to-Speech, Dialogflow, Cloud Translation, Cloud Natural Language API, Vision API, and Cloud Inference API), AutoML, AutoML Tables, or AI Platform. Google AI Platform Notebooks can be used as a development environment for data scientists, and Google AI Platform Training can be used to run training and scoring workloads at scale.

The following table shows Oracle ML functions.

Oracle	BigQuery
CLASSIFIER	Check BigQuery ML for ML classifier and regression options.
CLUSTER_DETAILS	
CLUSTER_DISTANCE	
CLUSTER_ID	
CLUSTER_PROBABILITY	
CLUSTER_SET	
PREDICTION	
PREDICTION_BOUNDS	

PREDICTION_COST	
PREDICTION_DETAILS	
PREDICTION_PROBABILITY	
PREDICTION_SET	

Security functions

The following table shows the functions for identifying the user in Oracle and BigQuery.

Oracle	BigQuery
UID	SESSION_USER
USER/SESSION_USER/CURRENT_USER	SESSION_USER()

Set/array functions

The following table shows set/array functions in Oracle and their equivalents in BigQuery.

Oracle	BigQuery
MULTISET	ARRAY_AGG
MULTISET EXCEPT	ARRAY_AGG([DISTINCT] expression)
MULTISET INTERSECT	ARRAY_AGG([DISTINCT])
MULTISET UNION	ARRAY_AGG

Window functions

The following table shows window functions in Oracle and their equivalents in BigQuery.

Oracle	BigQuery
LAG	LAG (value_expression[, offset [, default_expression]])
LEAD	LEAD (value_expression[, offset [, default_expression]])

Hierarchical or recursive queries

Hierarchical or recursive queries are not used in BigQuery. If the depth of the hierarchy is known, similar functionality can be achieved with joins, as illustrated in the following example. Another solution would be to utilize the BigQuery Storage API and Spark.

```
select
  array(
    select e.update.element
    union all
    select c1 from e.update.element.child as c1
    union all
    select c2 from e.update.element.child as c1, c1.child as c2
    union all
    select c3 from e.update.element.child as c1, c1.child as c2,
c2.child as c3
    union all
    select c4 from e.update.element.child as c1, c1.child as c2,
c2.child as c3, c3.child as c4
    union all
    select c5 from e.update.element.child as c1, c1.child as c2,
c2.child as c3, c3.child as c4, c4.child as c5
  ) as flattened,
  e as event
from t, t.events as e
```

The following table shows hierarchical functions in Oracle.

Oracle	BigQuery
DEPTH	Hierarchical queries are not used in BigQuery.
PATH	
SYS_CONNECT_BY_PATH	

UTL functions

The UTL_FILE package is mainly used for reading and writing the operating system files from PL/SQL. Cloud Storage can be used for any kind of raw file staging. External tables and BigQuery load and export should be used to read and write files from and to Cloud Storage. You can find details in the introduction to external data sources in BigQuery.

Spatial and GIS functions

You can use BigQuery GIS to replace spatial functionality. There are SDO_* functions and types in Oracle such as SDO_GEOM_KEY, SDO_GEOM_MBR, SDO_GEOM_MMB. These functions are used for spatial analysis. You can use BigQuery GIS to do spatial analysis.

DML syntax

This section addresses differences in data management language syntax between Oracle and BigQuery.

INSERT statement

Most Oracle INSERT statements are compatible with BigQuery. The following table shows exceptions.

DML scripts in BigQuery have slightly different consistency semantics than the equivalent statements in Oracle. For an overview of snapshot isolation and session and transaction handling, see the CREATE [UNIQUE] INDEX section elsewhere in this document.

<table>
<tr><th>Oracle</th><th>BigQuery</th></tr>
<tr><td><code>INSERT INTO table VALUES (...);</code></td><td><code>INSERT INTO table (...) VALUES (...);</code>

Oracle offers a DEFAULT keyword for non-nullable columns.

Note: In BigQuery, omitting column names in the INSERT statement only works if values for all columns in the target table are included in ascending order based on their ordinal positions.</td></tr>
<tr><td><code>INSERT INTO table VALUES (1,2,3);
INSERT INTO table VALUES (4,5,6);
INSERT INTO table VALUES (7,8,9);
INSERT ALL
   INTO table (col1, col2) VALUES ('val1_1', 'val1_2')
   INTO table (col1, col2) VALUES ('val2_1', 'val2_2')
   INTO table (col1, col2) VALUES ('val3_1', 'val3_2')
   .
   .
   .
SELECT 1 FROM DUAL;</code></td><td><code>INSERT INTO table VALUES (1,2,3),
                         (4,5,6),
                         (7,8,9);</code>

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 operation.
Combine multiple DML statements (including INSERT) using a MERGE statement.
Use CREATE TABLE ... AS SELECT to create and populate new tables.</td></tr>
</table>

UPDATE statement

Oracle UPDATE statements are mostly compatible with BigQuery, however, in BigQuery the UPDATE statement must have a WHERE clause.

As a best practice, you should prefer batch DML statements over multiple single UPDATE and INSERT statements. DML scripts in BigQuery have slightly different consistency semantics than equivalent statements in Oracle. For an overview on snapshot isolation and session and transaction handling, see the CREATE [UNIQUE] INDEX section in this document.

The following table shows Oracle UPDATE statements and BigQuery statements that accomplish the same tasks.

In BigQuery, the UPDATE statement must have a WHERE clause. For more information about UPDATE in BigQuery, see the BigQuery UPDATE examples in the DML documentation.

DELETE, TRUNCATE statements

The DELETE and TRUNCATE statements are both ways to remove rows from a table without affecting the table schema. TRUNCATE is not used in BigQuery. However, you can use DELETE statements to achieve the same effect.

In BigQuery, the DELETE statement must have a WHERE clause. For more information about DELETE in BigQuery, see the BigQuery DELETE examples in the DML documentation.

Oracle	BigQuery
DELETE *database.table*;	DELETE FROM *dataset.table* WHERE TRUE;

MERGE statement

The MERGE statement can combine INSERT, UPDATE, and DELETE operations into a single *upsert* statement and perform the operations atomically. The MERGE operation must match, at most, one source row for each target row. BigQuery and Oracle both follow ANSI syntax.

However, DML scripts in BigQuery have slightly different consistency semantics than the equivalent statements in Oracle.

DDL syntax

This section addresses differences in data definition language (DDL) syntax between Oracle and BigQuery.

`CREATE TABLE` statement

Most Oracle [`CREATE TABLE`] statements are compatible with BigQuery, except for the following constraints and syntax elements that are not used in BigQuery:

- `STORAGE`
- `TABLESPACE`
- `DEFAULT`
- `GENERATED ALWAYS AS`
- `ENCRYPT`
- `PRIMARY KEY (col, ...)`. See the `CREATE [UNIQUE] INDEX` section.
- `UNIQUE INDEX`. See the `CREATE [UNIQUE] INDEX` section.
- `CONSTRAINT..REFERENCES`
- `DEFAULT`
- `PARALLEL`
- `COMPRESS`

For more information about `CREATE TABLE` in BigQuery, see the BigQuery `CREATE` examples in the DML documentation.

Column attributes

Identity columns are introduced with Oracle 12c version, enabling auto-increment on a column. This is not used in BigQuery but can be achieved with the following batch way. For more information about surrogate keys and slowly changing dimensions (SCD), refer to the following guides:

- BigQuery Surrogate Keys
- BigQuery and surrogate keys: a practical approach

<table>
<tr><th>Oracle</th><th>BigQuery</th></tr>
<tr><td><pre>CREATE TABLE table (
  id   NUMBER GENERATED
ALWAYS AS IDENTITY,
  description VARCHAR2(30)
);</pre></td><td><pre>INSERT INTO dataset.table SELECT
  *,
  ROW_NUMBER() OVER () AS id
FROM dataset.table</pre></td></tr>
</table>

Column comments

Oracle uses `Comment` syntax to add comments on columns. This feature can be similarly implemented in BigQuery using the column description as shown in the following table.

Oracle	BigQuery
`Comment on column table is 'column desc';`	`CREATE TABLE dataset.table ( col1 STRING OPTIONS(description="column desc") );`

Temporary tables

Oracle supports temporary tables, which are often used to store intermediate results in scripts. Temporary tables are supported in BigQuery.

Oracle	BigQuery
`CREATE GLOBAL TEMPORARY TABLE temp_tab (x INTEGER, y VARCHAR2(50)) ON COMMIT DELETE ROWS; COMMIT;`	`CREATE TEMP TABLE temp_tab ( x INT64, y STRING ); DELETE FROM temp_tab WHERE TRUE;`

The following Oracle elements are not used in BigQuery:

- `ON COMMIT DELETE ROWS;`
- `ON COMMIT PRESERVE ROWS;`

There are also some other ways to emulate temporary tables in BigQuery:

- **Dataset TTL:** Create a dataset that has a short time to live (for example, one hour) so that any tables created in the dataset are effectively temporary (because they won't persist longer than the dataset's time to live). You can prefix all the table names in this dataset with `temp` to clearly denote that the tables are temporary.

- **Table TTL:** Create a table that has a table-specific short time to live using DDL statements similar to the following:

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

- **`WITH` clause:** If a temporary table is needed only within the same block, use a temporary result using a `WITH` statement or subquery.

CREATE SEQUENCE statement

Sequences are not used in BigQuery but can be achieved with the following batch way. For more information about surrogate keys and slowly changing dimensions (SCD), refer to the following guides:

- BigQuery Surrogate Keys
- BigQuery and surrogate keys: a practical approach

```
INSERT INTO dataset.table SELECT
  *,
  ROW_NUMBER() OVER () AS id
FROM dataset.table
```

CREATE VIEW statement

The following table shows equivalents between Oracle and BigQuery for the `CREATE VIEW` statement.

Oracle	BigQuery
`CREATE VIEW` *view_name* `AS SELECT ...`	`CREATE VIEW` *view_name* `AS SELECT ...`
`CREATE OR REPLACE VIEW` *view_name* `AS SELECT ...`	`CREATE OR REPLACE VIEW` *view_name* `AS SELECT ...`
Not supported.	`CREATE VIEW IF NOT EXISTS` *view_name* `OPTIONS(view_option_list) AS SELECT ...` Creates a new view only if the view doesn't exist in the specified dataset.

CREATE MATERIALIZED VIEW statement

In BigQuery, materialized view refresh operations are done automatically. There is no need to specify refresh options (for example, on commit or on schedule) in BigQuery. Materialized views in BigQuery are in beta as of September 2020. For more information about the BigQuery materialized view and its limitations, see the documentation.

In case the base table keeps changing by appends only, the query that uses the materialized view (whether the view is explicitly referenced or selected by the query optimizer) will scan all materialized views plus a delta in the base table since the last view refresh. This approach results in higher efficiency and lower costs.

On the contrary, if there were any updates (`DML UPDATE` / `DML MERGE`) or deletions (`DML DELETE`, truncation, partition expiration) in the base table since the last view refresh, the materialized view will not be scanned and hence query won't get any savings until the next view refresh. Any update or deletion in the base table invalidates the materialized view state.

Also, the data from the streaming buffer of the base table is not saved into the materialized view. The streaming buffer is still being scanned fully regardless of whether the materialized view is used.

The following table shows equivalents between Oracle and BigQuery for the `CREATE MATERIALIZED VIEW` statement.

Oracle	BigQuery

`CREATE MATERIALIZED VIEW` *`view_name`*
`REFRESH FAST NEXT sysdate + 7`
`AS SELECT … FROM TABLE_1`
	[`CREATE MATERIALIZED VIEW`](#)
	`view_name` `AS SELECT ...`

`CREATE [UNIQUE] INDEX` statement

This section describes approaches in BigQuery for how to create functionality similar to indexes in Oracle.

Indexing for performance

BigQuery doesn't need explicit indexes because it's a column-oriented database with query and storage optimization. BigQuery provides functionality such as partitioning and clustering as well as nested fields, which can increase query efficiency and performance by optimizing how data is stored.

Indexing for consistency (UNIQUE, PRIMARY INDEX)

In Oracle, a unique index can be used to prevent rows with non-unique keys in a table. If a process tries to insert or update data that has a value that's already in the index, the operation fails with an index violation.

Because BigQuery doesn't provide explicit indexes, a `MERGE` statement can be used instead to insert only unique records into a target table from a staging table while discarding duplicate records. However, there is no way to prevent a user with edit permissions from inserting a duplicate record.

To generate an error for duplicate records in BigQuery, you can use a `MERGE` statement from the staging table, as shown in the following example.

Oracle	BigQuery
`CREATE [UNIQUE] INDEX name;`	`` MERGE `prototype.FIN_MERGE` t USING `prototype.FIN_TEMP_IMPORT` m ON t.col1 = m.col1 AND t.col2 = m.col2 WHEN MATCHED THEN UPDATE SET t.col1 = ERROR(CONCAT('Encountered Error for ', m.col1, ' ', m.col2)) WHEN NOT MATCHED THEN INSERT (col1,col2,col3,col4,col5,col6,col7,col8) VALUES(col1,col2,col3,col4,col5,col6, CURRENT_TIMESTAMP(),CURRENT_TIMESTAMP()); ``

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

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

Locking

BigQuery doesn't have a lock mechanism like this and can run concurrent queries (up to your quota); only DML statements have certain concurrency limits and might require a table lock during execution in some scenarios.

Procedural SQL statements

This section describes how to convert procedural SQL statements used in stored procedures, functions, and triggers from Oracle to BigQuery.

`CREATE PROCEDURE` statement

Stored procedures are supported as part of BigQuery scripting.

Oracle	BigQuery
`CREATE PROCEDURE`	`CREATE PROCEDURE` Similar to Oracle, BigQuery supports `IN`, `OUT`, and `INOUT` argument modes. Other syntax specifications are not supported in BigQuery.
`CREATE OR REPLACE PROCEDURE`	`CREATE OR REPLACE PROCEDURE`
`CALL`	`CALL`

The sections that follow describe ways to convert existing Oracle procedural statements to BigQuery scripting statements that have similar functionality.

`CREATE TRIGGER` statement

Triggers are not used in BigQuery. Row-based application logic should be handled on the application layer. Trigger functionality can be achieved by using the ingestion tool, Pub/Sub, Cloud Functions, or a combination of tools during the ingestion time or by using regular scans.

Variable declaration and assignment

The following table shows Oracle `DECLARE` statements and their BigQuery equivalents.

Oracle	BigQuery

`DECLARE`
`   L_VAR NUMBER;`
`BEGIN`
`   L_VAR := 10 + 20;`
`END;`
	`DECLARE L_VAR int64;`
	`BEGIN`
	` SET L_VAR = 10 + 20;`
	` SELECT L_VAR;`
	`END`

`SET` *`var`* `=` *`value`*`;`	`SET` *`var`* `=` *`value`*`;`

Cursor declarations and operations

BigQuery does not support cursors, so the following statements are not used in BigQuery:

- `DECLARE` *`cursor_name`* `CURSOR [FOR | WITH] ...`
- `OPEN CUR_VAR FOR sql_str;`
- `OPEN cursor_name [USING var, ...];`
- `FETCH cursor_name INTO var, ...;`
- `CLOSE cursor_name;`

Dynamic SQL statements

The following table shows Oracle dynamic SQL statement and its BigQuery equivalent.

Oracle	BigQuery

`EXECUTE IMMEDIATE` *`sql_str`*
`[USING IN OUT [, ...]];`
	`EXECUTE IMMEDIATE sql_expression`
	`[ INTO variable[, ...] ]`
	`[ USING identifier[, ...] ];`

Flow-of-control statements

The following table shows Oracle flow-of-control statements and their BigQuery equivalents.

<table>
<tr><th>Oracle</th><th>BigQuery</th></tr>
<tr><td>IF condition THEN
  [if_statement_list]
[ELSE
  else_statement_list
]
END IF;</td><td>IF condition THEN
  [if_statement_list]
[ELSE
  else_statement_list
]
END IF;</td></tr>
<tr><td>SET SERVEROUTPUT ON;
DECLARE
x INTEGER DEFAULT 0;
y INTEGER DEFAULT 0;
BEGIN
LOOP
  IF x >= 10 THEN
    EXIT;
  ELSIF x >= 5 THEN
   y := 5;
  END IF;
  x := x + 1;
END LOOP;
dbms_output.put_line(x||','||y);
END;
/</td><td>DECLARE x INT64 DEFAULT 0;
DECLARE y INT64 DEFAULT 0;
LOOP
  IF x >= 10 THEN
    LEAVE;
  ELSE IF x >= 5 THEN
   SET y = 5;
   END IF;
  END IF;
  SET x = x + 1;
END LOOP;
SELECT x,y;</td></tr>
<tr><td>LOOP
  sql_statement_list
END LOOP;</td><td>LOOP
  sql_statement_list
END LOOP;</td></tr>
<tr><td>WHILE boolean_expression DO
  sql_statement_list
END WHILE;</td><td>WHILE boolean_expression DO
  sql_statement_list
END WHILE;</td></tr>
<tr><td>FOR LOOP</td><td>FOR LOOP is not used in BigQuery. Use other LOOP statements.</td></tr>
<tr><td>BREAK</td><td>BREAK</td></tr>
<tr><td>CONTINUE</td><td>CONTINUE</td></tr>
<tr><td>CONTINUE/EXIT WHEN</td><td>Use CONTINUE with IF condition.</td></tr>
<tr><td>GOTO</td><td>GOTO statement does not exist in BigQuery. Use IF condition.</td></tr>
</table>

Metadata and transaction SQL statements

<table>
<tr><th>Oracle</th><th>BigQuery</th></tr>
<tr><td>GATHER_STATS_JOB</td><td>Not used in BigQuery.</td></tr>
<tr><td>LOCK TABLE table_name IN
[SHARE/EXCLUSIVE] MODE NOWAIT;</td><td>Not used in BigQuery.</td></tr>
</table>

`Alter session set isolation_level=serializable; / SET TRANSACTION ...`	BigQuery uses snapshot isolation. For details, see Consistency guarantees in this document.
`EXPLAIN PLAN ...`	Not used in BigQuery. Similar features are the query plan explanation in the Cloud Console and the slot allocation, and in audit logging in Cloud Monitoring.
`SELECT * FROM DBA_[*];` (Oracle `DBA_/ALL_/V$` views)	`SELECT * FROM mydataset.INFORMATION_SCHEMA.TABLES;` For more information, see Introduction to BigQuery `INFORMATION_SCHEMA`.

`SELECT * FROM GV$SESSION;`
 	
`SELECT * FROM V$ACTIVE_SESSION_HISTORY;`
	BigQuery does not have the traditional session concept. You can see query jobs in the Cloud Console or export Cloud Monitoring audit logs to BigQuery and analyze BigQuery logs for analyzing jobs.

`START TRANSACTION;`
`LOCK TABLE table_A IN EXCLUSIVE MODE NOWAIT;`
`DELETE FROM table_A;`
`INSERT INTO table_A SELECT * FROM table_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.
	
	Using a query:
	`bq query --replace --destination_table table_A 'SELECT * FROM table_B';`
	
	Using a copy:
	`bq cp -f table_A table_B`

Multi-statement SQL blocks

Oracle supports transactions (sessions) and therefore supports statements separated by semicolons that are consistently executed together. In BigQuery, you can run a SQL block containing multiple statements separated by a semicolon; however, each SQL statement is atomic and is followed by an implicit commit.

Error codes and messages

Oracle error codes and BigQuery error codes are different. If your application logic is currently catching the errors, try to eliminate the source of the error because BigQuery does not return the same error codes.

Consistency guarantees and transaction isolation

Both Oracle 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 and updated values.

Transactions

Oracle provides read committed or serializable transaction isolation levels. Deadlocks are possible. Oracle insert append jobs run independently.

BigQuery helps ensure optimistic concurrency control (first to commit wins) with snapshot isolation, in which a query reads the last committed data before the query starts. This approach guarantees the same level of consistency on a per-row, per-mutation basis and across rows within the same DML statement, yet avoids deadlocks. In the case of multiple DML updates against the same table, BigQuery switches to pessimistic concurrency control. Load jobs can run completely independently and append to tables; however, BigQuery does not provide an explicit transaction boundary or session.

Rollback

Oracle supports rollbacks. Because there is no explicit transaction boundary in BigQuery, there is also no concept of an explicit rollback in BigQuery. The workarounds are table decorators or using `FOR SYSTEM_TIME AS OF`.

Database limits

Check BigQuery public documentation for the most recent quotas and limits. Many quotas for large-volume users can be raised by contacting the Cloud support team. The following table shows a comparison of the Oracle and BigQuery database limits.

Limit	Oracle	BigQuery
Tables per database	Unrestricted	Unrestricted
Columns per table	1,000	10,000
Maximum row size	Unlimited (depends on the column type)	100 MB
Column and table name length	If v12.2 >= 128 bytes Else 30 bytes	16,384 Unicode characters
Rows per table	Unlimited	Unlimited

Maximum SQL request length	Unlimited	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	Unlimited	10 MB (request) and 10 GB (response), or virtually unlimited if you use pagination or the Cloud Storage API.
Maximum number of concurrent sessions	Limited by the sessions or processes parameters	100 concurrent queries (can be raised with slot reservation), 300 concurrent API requests per user.
Maximum number of concurrent (fast) loads	Limited by the sessions or processes parameters	No concurrency limit; jobs are queued. 100,000 load jobs per project per day.

Other Oracle Database limits includes datatype limits, physical database limits, logical database limits, and process and runtime limits.