ShippingInstruction, Special Instructions, AllowPartialShipment, and LineItems.

```
{ "PONumber"             : 1600,
  "Reference"            : "ABULL-20140421",
  "Requestor"            : "Alexis Bull",
  "User"                 : "ABULL",
  "CostCenter"           : "A50",
  "ShippingInstructions" : { "name"    : "Alexis Bull",
                             "Address": { "street"  : "200 Sporting Green",
                                          "city"    : "South San Francisco",
                                          "state"   : "CA",
                                          "zipCode" : 99236,
                                          "country" : "United States of America" },
                             "Phone" : [ { "type" : "Office", "number" :
"909-555-7307" },
                                         { "type" : "Mobile", "number" :
"415-555-1234" } ] },
  "Special Instructions" : null,
  "AllowPartialShipment" : false,
  "LineItems"            : [ { "ItemNumber" : 1,
                               "Part"       : { "Description" : "One Magic Christmas",
                                                "UnitPrice"   : 19.95,
                                                "UPCCode"     : 13131092899 },
                               "Quantity"   : 9.0 },
                             { "ItemNumber" : 2,
                               "Part"       : { "Description" : "Lethal Weapon",
                                                "UnitPrice"   : 19.95,
                                                "UPCCode"     : 85391628927 },
                               "Quantity"   : 5.0 } ] }
```

In the preceding example, most properties have string values. PONumber, zipCode, ItemNumber, and Quantity have numeric values. Shipping Instructions and Address have objects as values. LineItems has an array as a value.

> **Note:**
>
> *Oracle XML DB Developer's Guide* for a more comprehensive overview of JSON

52

JSON and XML

Both JSON and XML are commonly used as data-interchange languages. Unlike relational data, both JSON data and XML data can be stored, indexed, and queried in the database without any schema that defines the data.

Because of its simple definition and features, JSON data is generally easier to generate, parse, and process than XML data. It is also easier for human beings to learn and to use. The following table describes further differences between JSON and XML.

> **See Also:**
>
> - EXPLAIN PLAN Statement
> - *Oracle Database SQL Tuning Guide* for more information about Full Table Scans
> - *Oracle Database SQL Tuning Guide* for more information about Index Range Scans

9.2 Representing Character Data

Table 9-1 summarizes the SQL data types that store character data.

Table 9-1 SQL Character Data Types

Data Types	Values Stored
CHAR	Fixed-length character literals
VARCHAR2	Variable-length character literals
NCHAR	Fixed-length Unicode character literals
NVARCHAR2	Variable-length Unicode character literals
CLOB	Single-byte and multibyte character strings of up to (4 gigabytes - 1) * (the value obtained from DBMS_LOB.GETCHUNKSIZE)
NCLOB	Single-byte and multibyte Unicode character strings of up to (4 gigabytes - 1) * (the value obtained from DBMS_LOB.GETCHUNKSIZE)
LONG	Variable-length character data of up to 2 gigabytes - 1. Provided only for backward compatibility.

52

> **Note:**
>
> Do not use the VARCHAR data type. Use the VARCHAR2 data type instead. Although the VARCHAR data type is currently synonymous with VARCHAR2, the VARCHAR data type is scheduled to be redefined as a separate data type used for variable-length character strings compared with different comparison semantics.

When choosing between CHAR and VARCHAR2, consider:

- Space usage

  Oracle Database blank-pads values stored in CHAR columns but not values stored in VARCHAR2 columns. Therefore, VARCHAR2 columns use space more efficiently than CHAR columns.

- Performance

  Because of the blank-padding difference, a full table scan on a large table containing VARCHAR2 columns might read fewer data blocks than a full table scan

on a table containing the same data stored in CHAR columns. If your application often performs full table scans on large tables containing character data, then you might be able to improve performance by storing data in VARCHAR2 columns rather than in CHAR columns.

- Comparison semantics

  When you need ANSI compatibility in comparison semantics, use the CHAR data type. When trailing blanks are important in string comparisons, use the VARCHAR2 data type.

For a client/server application, if the character set on the client side differs from the character set on the server side, then Oracle Database converts CHAR, VARCHAR2, and LONG data from the database character set (determined by the NLS_LANGUAGE parameter) to the character set defined for the user session.

See Also:

- *Oracle Database SQL Language Reference* for more information about comparison semantics for these data types
- Large Objects (LOBs) for more information about CLOB and NCLOB data types
- LONG and LONG RAW Data Types for more information about LONG data type

52

9.3 Representing Numeric Data

The SQL data types that store numeric data are NUMBER, BINARY_FLOAT, and BINARY_DOUBLE.

The NUMBER data type stores real numbers in either a fixed-point or floating-point format. NUMBER offers up to 38 decimal digits of precision. In a NUMBER column, you can store positive and negative numbers of magnitude 1 x 10^{-130} through 9.99 x10^{125}, and 0. All Oracle Database platforms support NUMBER values.

The BINARY_FLOAT and BINARY_DOUBLE data types store floating-point numbers in the single-precision (32-bit) IEEE 754 format and the double-precision (64-bit) IEEE 754 format, respectively. High-precision values use less space when stored as BINARY_FLOAT and BINARY_DOUBLE than when stored as NUMBER. Arithmetic operations on floating-point numbers are usually faster for BINARY_FLOAT and BINARY_DOUBLE values than for NUMBER values.

In client interfaces that Oracle Database supports, arithmetic operations on BINARY_FLOAT and BINARY_DOUBLE values are performed by the native instruction set that the hardware vendor supplies. The term **native floating-point data type** includes BINARY_FLOAT and BINARY_DOUBLE data types and all implementations of these types in supported client interfaces.

Native floating-point data types conform substantially with the Institute of Electrical and Electronics Engineers (IEEE) Standard for Binary Floating-Point Arithmetic, IEEE Standard 754-1985 (IEEE754).

- Divide by zero
- Underflow
- Overflow

However, Oracle Database does not raise these exceptions for native floating-point data types. Generally, operations that raise exceptions produce the values described in Table 9-6.

Table 9-6 Values Resulting from Exceptions

Exception	Value
Underflow	`0`
Overflow	`-INF`, `+INF`
Invalid Operation	`NaN`
Divide by Zero	`-INF`, `+INF`, `NaN`
Inexact	Any value – rounding was performed

9.3.7 Client Interfaces for Native Floating-Point Data Types

Oracle Database supports native floating-point data types in these client interfaces:

- SQL and PL/SQL

  Support for `BINARY_FLOAT` and `BINARY_DOUBLE` includes their use as attributes of Abstract Data Types (ADTs), which you create with the SQL statement `CREATE TYPE` (fully described in *Oracle Database PL/SQL Language Reference*).

- Oracle Call Interface (OCI)

  For information about using `BINARY_FLOAT` and `BINARY_DOUBLE` with OCI, see *Oracle Call Interface Programmer's Guide*.

- Oracle C++ Call Interface (OCCI)

  For information about using `BINARY_FLOAT` with OCCI, see *Oracle C++ Call Interface Programmer's Guide*.

  For information about using `BINARY_DOUBLE` with OCCI, see *Oracle C++ Call Interface Programmer's Guide*.

- Pro*C/C++ precompiler

  To use `BINARY_FLOAT` and `BINARY_DOUBLE`, set the Pro*C/C++ precompiler command line option `NATIVE_TYPES` to `YES` when you compile your application. For information about the `NATIVE_TYPES` option, see *Pro*C/C++ Programmer's Guide*.

- Oracle JDBC

  For information about using `BINARY_FLOAT` and `BINARY_DOUBLE` with Oracle JDBC, see *Oracle Database JDBC Developer's Guide*.

52

9.4 Representing Date and Time Data

Oracle Database stores `DATE` and `TIMESTAMP` (**datetime**) data in a binary format that represents the century, year, month, day, hour, minute, second, and optionally, fractional seconds and timezones.

Table 9-9 Large Objects (LOBs)

Data Type	Description
BLOB	**Binary large object** Stores any kind of data in binary format. Typically used for multimedia data such as images, audio, and video.
CLOB	**Character large object** Stores string data in the database character set format. Used for large strings or documents that use the database character set exclusively.
NCLOB	**National character large object** Stores string data in National Character Set format. Used for large strings or documents in the National Character Set.
BFILE	**External large object** Stores a binary file outside the database in the host operating system file system. Applications have read-only access to BFILEs. Used for static data that applications do not manipulate, such as image data. Any kind of data (that is, any operating system file) can be stored in a BFILE. For example, you can store character data in a BFILE and then load the BFILE data into a CLOB, specifying the character set when loading.

An instance of type BLOB, CLOB, or NCLOB can be either **temporary** (declared in the scope of your application) or **persistent** (created and stored in the database).

> **See Also:**
>
> - *Oracle Database SecureFiles and Large Objects Developer's Guide* for information about using LOBs in application development
> - *Oracle Database SQL Language Reference* for more information about LOB functions

9.5.2.2 LONG and LONG RAW Data Types

> **Note:**
>
> Oracle supports the LONG and LONG RAW data types for backward compatibility, but strongly recommends that you convert LONG columns to LOB columns and LONG RAW columns to BLOB columns.

LONG columns store variable-length character strings containing up to 2 gigabytes - 1 bytes. .

The LONG RAW (and RAW) data types store data that is not to be explicitly converted by Oracle Database when moving data between different systems. These data types are intended for binary data or byte strings.