Porting from Oracle to PostgreSQL

by Paulo Merson February/2002

If you are starting to use PostgreSQL or you will migrate from Oracle database server, I hope this document helps. If you have Java applications and use JDBC, the "Data types and JDBC" section will be particularly useful.

Oracle and PostgreSQL both conform to standard SQL. However, they contain several extensions and implementation details that differentiate one from the other. The most important differences are listed in this document.

If you have comments about this document, please email me: pmerson@cs.cmu.edu.

1. SQL Syntax, Functions, Sequences, Etc.

Oracle	**PostgreSQL**
`select sysdate from dual`	`select 'now'::datetime` There is no "dual" table Unlike other RDBMS, PostgreSQL allows a "select" without the "from" clause. This use does not affect portability because the syntax to get current time is already DBMS specific.

`CREATE SEQUENCE` *`seqname`* `[ INCREMENT BY` *`integer`* `]`
`[ MINVALUE` *`integer`* `] [ MAXVALUE` *`integer`* `]`
`[ START WITH` *`integer`* `] [ CACHE` *`integer`* `] [ CYCLE | NOCYCLE ]`
 	
Oracle's "create sequence" has other arguments not listed here and not supported by PostgreSQL, but the main difference is the need of 'by' and "with" after "increment" and "start".
 	
If you don't specify "MAXVALUE" or if you use the parameter "NOMAXVALUE", then the actual limit is 10^{27}.
	`CREATE SEQUENCE` *`seqname`* `[ INCREMENT` *`increment`* `]`
	`[ MINVALUE` *`minvalue`* `] [ MAXVALUE` *`maxvalue`* `]`
	`[ START` *`start`* `] [ CACHE` *`cache`* `] [ CYCLE ]`
	
	If you don't specify MAXVALUE, then the maximum value is 2147483647 for ascending sequences.

Oracle	**PostgreSQL**

To return the current value and increment the counter:
`sequence_name.nextval;`
Possible usage in a select statement:
`select sequence_name.nextval from dual;`
	To return the current value and increment the counter:
	`nextval(‘sequence_name’);`
	Possible usage in a select statement
	`select nextval(‘sequence_name’);`
	Note that unlike other RDBMS, PostgreSQL allows a select without the ‘from’ clause. This use does not affect portability because the sequence syntax is already DBMS specific.

`SELECT product_id,`
`       DECODE (warehouse_id, 1, ’Southlake’,`
`                             2, ’San Francisco’,`
`                             3, ’New Jersey’,`
`                             4, ’Seattle’,`
`                             ’Non-domestic’)`
`       quantity_on_hand`
`FROM inventories`
	`SELECT a,`
	`       CASE WHEN a=1 THEN 'one'`
	`            WHEN a=2 THEN 'two'`
	`            ELSE 'other'`
	`       END`
	`FROM test`

`select employeeid,`
`       NVL(hire_date, sysdate)`
`from employee`
`where employeeid = 10;`
Oracle also has a “coalesce” function that is a generalization of the commonly used NVL function.
	`select employeeid,`
	`       coalesce(hire_date, 'now'::datetime)`
	`from employee`
	`where employeeid = 10;`

Outer join – “(+)”	Doesn’t support outer join. The workaround is to use a union.
Hierarchical queries – “CONNECT BY”	Nothing similar

`SELECT product_id FROM inventories`
`MINUS`
`SELECT product_id FROM order_items;`
	`SELECT product_id FROM inventories`
	`EXCEPT`
	`SELECT product_id FROM order_items;`

`select unique col1, col2 from table1` In Oracle “distinct” and “unique” keywords are synonymous in the select statement.	`select distinct col1, col2 from table1` PostgreSQL doesn’t allow “select unique”.

Oracle	**PostgreSQL**

Oracle relational operators may have a space between the characters. For example, the following select will work:
`select id,name from employee where id >     = 10;`
`// There are spaces and tabs between “>” and “=”`
	PostgreSQL relational operators doesn’t allow spaces, the characters that compound an operator must be consecutive when the command is parsed:
	`select id,name from employee where id >= 10;`

To get the remainder of the division of 10 by 4 (modulo) use the “mod” function: `select mod(10,4) from dual;`	To get the remainder of the division of 10 by 4 (modulo) use the “%” operator. (And PostgreSQL has many other arithmetic operators.) `select 10 % 4;`

The “ROWNUM” pseudo-column returns a number indicating the order in which Oracle selects the row.
ROWNUM can be used to limit the number of rows returned by a query, for example:
`select * from employees where rownum < 10 order by name;`
ROWNUM can be used in the projection as one of the values returned by the query (first line has value 1, second line value 2, and so on):
`select rownum, name from employees order by name;`
	There isn’t anything equivalent to Oracle ROWNUM.
	However, you can limit the number of rows returned by a query using the “LIMIT” clause:
	`select * from employees order by name limit 10;`
	In some cases, it’s possible that the pseudo-column OID may substitute ROWNUM, although they have different behavior. OID is a unique identifier of each line per table, while ROWNUM is always 1, 2, …, N for each different query.
	`select oid, name from employees order by name;`
	And the query that uses ROWNUM can have join tables. If your select is a join you’ll have a different OID for each table, because each one has an OID column.

2. Database Server General Characteristics

Oracle	**PostgreSQL**
A view can be “updatable” if some conditions are satisfied.	Views are read only.

Oracle	PostgreSQL
Transactions are initiated implicitly. By default, the "auto-commit" behavior is disabled.	BEGIN – initiates a transaction disabling the default behavior, which is "auto-commit" enabled, i.e., a commit is performed after each user statement. In Java, we need to write: con.setAutoCommit(false)
Tables, views and other objects can be logically grouped in "schemas". A schema usually maps to the user name of the user that created the objects (owner). Thus, a table can be referenced in a statement as "schemaName.tableName". For example: `select * from mySchema.myTable;`	There is no schema support, but it's planned for a future version. The alternative is to use separate databases: You have to connect to the specific database and use that connection to execute your SQL command. However, if you have a SQL statement that uses tables in different (Oracle) schemas, you cannot use separate (PostgreSQL) databases; there is no direct workaround, you'll need to rewrite the code.
Interactive command prompt tool: SQL*Plus	Interactive command prompt tool: psql
Oracle permissions are granted/revoked to/from users or *roles*. You can create roles and *grant*/*revoke* roles to/from users. But you can "create" and "alter" groups to insert and remove users.	PostgreSQL permissions are granted/revoked to/from users or *groups*. You can create groups and then alter the groups *inserting*/*removing* users.
By default, a password is always required to connect to the database.	By default, you can connect to the database simply by specifying the database name, no user ID and password are required. You should follow the instructions in the Administrator's Guide to configure the pg_hba.conf file in order to use password authentication.

3. Data Types and JDBC

Oracle	JDBC*	PostgreSQL	JDBC
NUMBER(p) *where p is the precision, i.e., the number of digits*	getByte getShort getInt getLong	SMALLINT - *2 bytes*	getShort
		INTEGER - *4 bytes*	getInt
		BIGINT - *8 bytes*	getLong
NUMBER(p,s) *where p is the total number of digits and s is the number of digits to the right of the decimal point*	getDouble getBigDecimal	NUMERIC(p,s)	getBigDecimal
		REAL – *4 bytes, 6 decimal places*	getDouble
		DOUBLE PRECISION – *8 bytes, 15 decimal places*	getDouble
Nothing similar		SERIAL – *0 to +2147483647, typically used to create unique identifiers. Generates an implicit sequence that is incremented when a line is inserted in the table.*	getInt
VARCHAR2(size) *where maximum size is 4000*	getString	CHARACTER VARYING(*n*) *where maximum n is 1 GB* VARCHAR(n) is an alias	getString
CHAR(size) *where maximum size is 2000*	getString	CHARACTER(n) *where maximum n is 1 GB* CHAR(n) is an alias *It's suggested that you use TEXT if n > 10 MB*	getString
LONG - *Character data of variable length up to 2 GB*	getString	TEXT – *variable length up to 1 GB* *It's suggested that you use TEXT if n > 10 MB*	getString
DATE – *holds date and time*	getDate getTime getTimestamp	TIMESTAMP You still can use getDate to read a TIMESTAMP column, but you will loose the time portion of the data.	getTimestamp
TIMESTAMP	oracle.sql. getTIMESTAMP		
Nothing similar		DATE – *holds only the date (resolution is one day)*	getDate
Nothing similar		TIME – *holds only the time (00:00:00.00 – 23:59:59.99)*	getTime

Oracle	JDBC*	PostgreSQL	JDBC
RAW(size) – *binary data of length size bytes (max 2000)* LONG RAW – *binary data or variable length up to 2GB*	getBytes	BYTEA	getBytes
Nothing similar		BIT(n) – *fixed length string of 1's and 0's* BIT VARYING(n) – *variable length string of 1's and 0's*	(?)
CLOB – *character large object (max 4GB)*	getClob	TEXT *(max 1GB)*	getString
BLOB – *binary large object (max 4GB)*	getBlob	BYTEA *(max 1GB)* *BYTEA is not documented in PostgreSQL 7.1 but it's fully implemented; Jdbc 7.2-1.2 is required though in order to use getBytes and setBytes.* *Besides TEXT and BYTEA, PostgreSQL supports large objects as separate files. They are stored in a separate table in a special format, and are referred to from regular tables by an OID value. More information:* *http://www.postgresql.org/idocs/index.php?largeobjects.html* *http://www.postgresql.org/idocs/index.php?jdbc-lo.html*	getBytes
ROWID	oracle.sql.getROWID	*Nothing similar*	
Nothing similar *Typically,* char(1) *is used to store a value that is translated to Boolean in the application logic.* *If you store '0' and '1' in a varchar2(1) or char column, then the jdbc driver can correctly interpret these values as boolean false an true respectively using ResultSet.getBoolean and PreparedStatement.setBoolean*		BOOLEAN – *can have the value TRUE, FALSE or NULL* *If you store '0' and '1' in a varchar(1) or char(1) column, then the jdbc driver can correctly interpret these values as boolean false an true respectively using ResultSet.getBoolean. However, PreparedStatement.setBoolean simply does not work.* *If you use PostgreSQL BOOLEAN, then your Java code can use getBoolean and setBoolean.*	getBoolean
Oracle Spatial features	*?*	*Geometric data types:* POINT, LINE, CIRCLE, *etc.*	org.postgresql.geometric.*

Oracle	JDBC*	PostgreSQL	JDBC
Nothing similar		*Network address data types:* `INET, MACADDR, CIDR.`	*(?)*

* JDBC note: Typically `PreparedStatement.set`*`Xxx`* is used to set the value of arguments (or "host" variables) inside SQL statements. And there is a correspondent `ResultSet.get`*`Xxx`* method to read the value returned by a query into java variables. Each `get/setXXX` method has a specific Java data type or class associated to it (e.g. `set/getInt` deals with `int`; `set/getDate` deals with `java.sql.Date`, etc.). Further, you can use different methods to read the same database data type, but usually there is a *recommended* method. For example, a BIGINT column can be read with `getShort`, `getInt`, `getLong`, `getDouble`, etc., but the recommended method is `getLong`.

So, to indicate the proper way to use each data type in Java I simply listed the recommended `getXxx` JDBC method.

4. Other Considerations:

- The set of operators and SQL functions is very similar though Oracle has a richer set. For example, both DBMS have the concatenation operator "||", as well as `substr`, `upper`, `to_char` and other functions with the same syntax. However, any Oracle function that is being used must have its syntax compared to the equivalent function in PostgreSQL, if such exists.
- PostgreSQL lacks the ability to query across multiple databases.
- PostgreSQL's PL/pgSQL is similar to Oracle PL/SQL and can be used to write stored functions. PostgreSQL doesn't have packages or procedures (only functions). More about this: http://www.postgresql.org/idocs/index.php?plpgsql-porting.html
- Both DBMS have triggers and the `create trigger` statement is similar, but the code executed by the trigger for PostgreSQL must be in a stored function written by the user, while in Oracle you have the option of writing the code in a PL/SQL block in the `create trigger` statement. PostgreSQL has yet an additional resource called the "rule system" that allows the definition of business logic that is executed upon an event.
- The `create table` statement is similar in both DBMS. One noticeable difference is that PostgreSQL doesn't have `pctfree`, `pctused`, `inittrans`, and `maxtrans` clauses. They also differ in the `create database` statement, mainly in the arguments and clauses that specify storage details.

5. References:

- Oracle 9i documentation - http://download-east.oracle.com/otndoc/oracle9i/901_doc/nav/docindex.htm
- PostgreSQL documentation - http://www.postgresql.org/idocs/
- “Oracle to Postgre Conversion” - http://openacs.org/doc/openacs/html/oracle-to-pg-porting.html
- “PostgreSQL JDBC 2.0 compliance” - http://lab.applinet.nl/postgresql-jdbc/
- An important source of information is the PostgreSQL mailing lists: http://archives.postgresql.org/