Overview

Azure PostgreSQL is a general purpose and object-relational database management system,the most advanced open source database system. PostgreSQL was designed to run on UNIX-like platforms. However, PostgreSQL was then also designed to be portable so that it could run on various platforms such as Mac OS X, Solaris, and Windows. PostgreSQL is free and open source software. Its source code is available under PostgreSQL license, a liberal open source license. You are free to use, modify and distribute PostgreSQL in any form PostgreSQL requires very minimum maintained efforts because of its stability. Therefore, if you develop applications based on PostgreSQL, the total cost of ownership is low in comparison with other database management systems. PostgreSQL is designed to be extensible. In PostgreSQL, you can define your own data types, index types, functional languages, etc. If you do not like any part of the system, you can always develop a custom plugin to enhance it to meet your requirements e.g., adding a new optimizer.

Scan Result

Assessment Name	dbassessment
Name of Database	
Size (mb)	0.12 MB
Server Version/Edition	Oracle Database 19c Enterprise Edition Release 19.0.0.0.0

Recommendation Result

Migration level

Migration with code rewrite and a human-days cost up to 5 days

Technical level

difficult: stored functions and/or triggers with code rewriting

93 hrs

TOTAL EFFORTS

Azure PostgreSQL

Gen 5, 4 vCore USD $ 255.792

Object	Recommendations
GLOBAL TEMPORARY TABLE	**Issue Detail** Global temporary table are not supported by PostgreSQL and will not be exported. You will have to rewrite some application code to match the PostgreSQL temporary table behavior. **Recommendation** PostgreSQL doesn't support Global Temporary tables. Only local temporary table are available in PostgreSQL in which values can be persist within a session only. This will not serve the purpose of Global temporary table where table need to be shared among multiple session and each session has its own set of data which are not overlapped by another session. So either replace all Global Temporary Tables with azure PostgreSQL views or convert all the global temporary table as a regular PostgreSQL table to share the among multiple sessions.

Actual Object Count	Estimated Effort
37	1344 minutes

Help Url

No Record Found

INDEX

Issue Detail

48 index(es) are concerned by the export, others are automatically generated and will do so on PostgreSQL. Bitmap will be exported as btree_gin index(es) and hash index(es) will be exported as b-tree index(es) if any. Domain index are exported as b-tree but commented to be edited to mainly use FTS. Cluster, bitmap join and IOT indexes will not be exported at all. Reverse indexes are not exported too, you may use a trigram-based index (see pg_trgm) or a reverse() function based index and search. Use 'varchar_pattern_ops', 'text_pattern_ops' or 'bpchar_pattern_ops' operators in your indexes to improve search with the LIKE operator respectively into varchar, text or char columns.

Recommendation

PostgreSQL has several index types: B-tree, Hash, GiST, GIN, and BRIN.

Each index type uses a different storage structure and algorithm to cope with different kinds of queries.

B-tree is a self-balancing tree that maintains sorted data and allows searches, insertions, deletions, and sequential access in logarithmic time.

By default, B-Tree index is created when CREATE INDEX statement is executed.

Lets see the some comparison of oracle and PostgreSQL index creation and alteration:

Crete Oracle Index :

```
CREATE UNIQUE INDEX IDX_Cust_ID ON Customer(Cust_ID);
```

PostgreSQL CREATE Index:

```
CREATE UNIQUE INDEX IDX_Cust_ID ON Customer(Cust_ID)
```

Oracle Alter Index:

```
ALTER INDEX IDX_Cust_ID RENAME TO IDX_Cust_ID_NEW;
```

PostgreSQL Alter Index;

```
ALTER INDEX IDX_Cust_ID RENAME TO IDX_Cust_ID_NEW;
```

Oracle REBUILD INDEX:

```
ALTER INDEX IDX_Cust_ID REBUILD;
```

PostgreSQL REINDEX (REBUILD) INDEX

```
REINDEX INDEX IDX_Cust_ID;
```

Create Oracle Composite INDEX

```
CREATE INDEX IDX_Cust_COMPI ON Customer (Customer_NAME, PHONE_NUMBER);
```

Create Multicolumn Index PostgreSQL

```
CREATE INDEX IDX_Cust_COMPI ON Customer (Customer_NAME, PHONE_NUMBER);
```

There are few index type which are not supported directly in PostgreSQL but the similar functionality can be achieved in other way.

1. Oracle BITMAP Index

```
CREATE BITMAP INDEX IDX_BITMAP_Cust_Cat ON Customer(Category);
```

We can use BRIN (Block Range Index) index instead for BITMAP index. Records are grouped into block range with min/max summaries

CREATE INDEX IDX_BRIN_Cust ON Customer USING BRIN(Category);

Oracle Function Based index can be replaced with PostgreSQL Expression index.

Ex: CREATE INDEX idx_ic_last_name ON Customer(LOWER(last_name));

Actual Object Count	Estimated Effort
88	109 minutes

Help Url

https://www.postgresql.org/docs/current/textsearch-tables.html

INDEX PARTITION

Issue Detail

Only local indexes partition are exported, they are build on the column used for the partitioning.

Recommendation

In oracle Local and Global Indexes are used for Partitioned Tables.

A local index on a partitioned table is created where the index is partitioned in exactly the same manner as the underlying partitioned table.

That is, the local index inherits the partitioning method of the table. This is known as equi-partitioning.

Create oracle Local Index :

```
CREATE INDEX test_idx ON EXP_PART(id, item_id)
  LOCAL
 (partition test_idx_1,
  partition test_idx_2,
  partition test_idx_3,
  partition test_idx_4);
```

Create oracle Global Index :

A global partitioned index is an index on a partitioned or non-partitioned table that is partitioned independently,

i.e. using a different partitioning key from the table. Global-partitioned indexes can be range or hash partitioned.

Global partitioned indexes are more difficult to maintain than local indexes. However, they do offer a more efficient access method to any individual record.

```
CREATE UNIQUE INDEX EXP_PART_IDX ON EXP_PART(id, item_id)
  GLOBAL PARTITION BY RANGE (id, item_id)
```

(partition EXP_PART_IDX_1 values less than (20, 200),

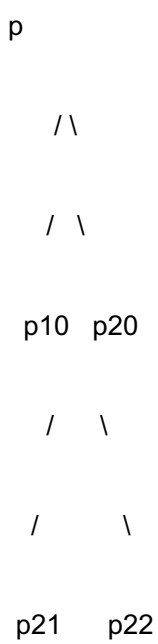
partition EXP_PART_IDX_2 values less than (maxvalue, maxvalue));

PostreSQL :

The implementation of partitioning in PostgreSQL (“TableInheritance“) includes the use of a Parent Table with Child Tables used as the table partitions.

Any index created on a partition table will be created as well on each existing child tables. Any future partition added will gain the same index as well.

For example, taking this layer of partitioned tables:



Example :

```
CREATE TABLE p (a int) PARTITION BY RANGE (a);

CREATE TABLE p10 PARTITION OF p FOR VALUES FROM (0) TO (10)  PARTITION BY RANGE(a);

CREATE TABLE p20 PARTITION OF p FOR VALUES FROM (10) TO (20) PARTITION BY RANGE(a);

CREATE TABLE p21 PARTITION OF p20 FOR VALUES FROM (10) TO (11);

CREATE TABLE p22 PARTITION OF p20 FOR VALUES FROM (11) TO (12);
```

The creation of an index gets executed to all partitions. So, for example creating an index on ‘p’ results in the same being created down to all partitions.

```
CREATE INDEX pi ON p(a)
```

Actual Object Count	Estimated Effort
15	0 minutes

Help Url

No Record Found

PACKAGE BODY

Issue Detail

Total size of package code: 326580 bytes. Number of procedures and functions found inside those packages: 77.

Impacted Object	Impacted Type	Estimated Effort
benchmarking.uoft_benchmarking	Procedure	92 minutes
dia_data_etl_common.create_gsc_claim_hist_job	Procedure	62 minutes
dia_data_etl_common.create_gsc_membership_job	Procedure	16 minutes
dia_data_etl_common.create_hbmp_claim_hist_job	Procedure	46 minutes
dia_data_etl_common.create_hbmp_revenue_hist_job	Procedure	56 minutes
dia_data_etl_common.create_part_offg_job	Procedure	16 minutes
dia_data_etl_common.create_term_cnt_job	Procedure	5 minutes
dia_data_etl_common.end_workflow_job_n	Procedure	24 minutes
dia_data_etl_common.end_workflow_job_old	Procedure	6 minutes
dia_data_etl_common.get_run_dur_seconds	Procedure	8 minutes
dia_data_etl_common.init_job_with_param	Procedure	13 minutes
dia_data_etl_common.update_run_duration	Procedure	12 minutes
dia_data_etl_ext.load_client_onboard_info	Procedure	6 minutes
dia_data_etl_ext.load_revenue_daily	Procedure	136 minutes
dia_data_mart_common.create_gsc_membership_job	Procedure	16 minutes
dia_data_mart_common.create_new_wf_job_with_param	Procedure	13 minutes
dia_data_mart_common.get_run_dur_seconds	Procedure	8 minutes
dia_data_mart_common.update_run_duration	Procedure	12 minutes
dia_hbmp_plan_insight.load_hbmp_plan_insights	Procedure	49 minutes
dia_hbm_plan_insight.load_hbm_plus_plan_insights	Procedure	49 minutes
dia_individual_etl.ind_app_addon	Procedure	57 minutes
dia_individual_etl.ind_claims_dwp	Procedure	83 minutes
dia_individual_etl.ind_inforce	Procedure	66 minutes
dia_individual_etl.ind_plan_ref	Procedure	36 minutes
dia_individual_etl.ind_revenue_dwp	Procedure	75 minutes
dia_plan_design_util.fn_check_ref_tbl_data_quality	Procedure	101 minutes
dia_plan_design_util.fn_get_accum_start_date	Procedure	83 minutes

dia_plan_design_util.fn_get_bnft_catg_full_path	Procedure	24 minutes
dia_plan_design_util.get_catg_bnft_hierarchy	Procedure	86 minutes
dia_plan_design_util.initialize_mxrl_ref_tables	Procedure	7 minutes
dia_plan_design_util.is_check_level_break	Procedure	43 minutes
dia_plan_design_util.load_max_rules_accum_hist	Procedure	40 minutes
dia_plan_design_util.load_max_rules_acum_summary	Procedure	16 minutes
dia_plan_design_util.load_max_rules_param	Procedure	88 minutes
load_mega_rept_data.create_alteryx_wf_job	Procedure	12 minutes
load_mega_rept_data.create_claim_hist_wf_job	Procedure	46 minutes
load_mega_rept_data.create_revenue_hist_wf_job	Procedure	56 minutes
load_mega_rept_data.get_run_dur_seconds	Procedure	8 minutes
load_mega_rept_data.load_ct_ace_psr_broker	Procedure	265 minutes
load_mega_rept_data.load_pl_bnft_ref_monthly	Procedure	52 minutes
load_mega_rept_data.load_pl_provider_ref_monthly	Procedure	20 minutes
load_mega_rept_data.purge_duplicate_revenu_hist	Procedure	43 minutes
load_mega_rept_data.purge_failed_claim_hist	Procedure	52 minutes
load_mega_rept_data.purge_failed_revenue_hist	Procedure	52 minutes
load_mega_rept_data.refresh_table	Procedure	84 minutes
load_mega_rept_data.update_bnft_ref_level	Procedure	60 minutes
load_mega_rept_data.update_run_duration	Procedure	12 minutes
plan_insights_etl.load_plan_insights_summ	Procedure	60 minutes
plan_insights_etl.load_pl_claims_summary	Procedure	96 minutes
plan_insights_etl.load_stg2	Procedure	44 minutes

Recommendation

Packages cannot be created in PostgreSQL, so define a schema with the same name as the package and define functions/Procedures that have a
relationship in the schema so that they are treated as a single group.

Let's consider a oracle package:

```
create or replace PACKAGE cust_sal AS
  PROCEDURE find_sal(c_id customers123.id%type);
END cust_sal;

create or replace PACKAGE BODY cust_sal AS
  PROCEDURE find_sal(c_id customers123.id%TYPE) IS
  c_sal customers123.salary%TYPE;
  BEGIN
    SELECT salary INTO c_sal
    FROM customers123
    WHERE id = c_id;
    dbms_output.put_line('Salary: '|| c_sal);
  END find_sal;
END cust_sal;
```

```
-- EXEC cust_sal.find_sal(1);
```

Workaround for package in PostgreSQL:

1. Create the schema same as package name

```
CREATE SCHEMA  cust_sal ;
```

2.Convert the oracle package function /procedure in PostgreSQL and execute in the above schema.

```
CREATE OR REPLACE PROCEDURE cust_sal.find_sal(C_ID DECIMAL(38,0))
AS
$$
 DECLARE
   C_SAL DECIMAL(18,2);
   BEGIN
      SELECT SALARY INTO C_SAL
      FROM CUSTOMERS123
      WHERE ID = C_ID;
      RAISE NOTICE '%',('SALARY: '|| C_SAL);
   END;
$$ LANGUAGE PLPGSQL;

-- call cust_sal.find_sal(1);
```

Actual Object Count	Estimated Effort
10	2855 minutes

Help Url

https://www.postgresql.org/docs/7.3/plpgsql-porting.html

PROCEDURE

Issue Detail

Total size of procedure code: 0 bytes.

Recommendation

CREATE PROCEDURE defines a new procedure. CREATE OR REPLACE PROCEDURE will either create a new procedure, or replace an existing definition. To be able to define a procedure, the user must have the USAGE privilege on the language.

If a schema name is included, then the procedure is created in the specified schema. Otherwise it is created in the current schema.

When CREATE OR REPLACE PROCEDURE is used to replace an existing procedure, the ownership and permissions of the procedure do not change. All other procedure properties are assigned the values specified or implied in the command. You must own the procedure to replace it (this includes being a member of the owning role).

The user that creates the procedure becomes the owner of the procedure.

To be able to create a procedure, you must have USAGE privilege on the argument types.

Oracle -

```
CREATE OR REPLACE PROCEDURE GET_CUSTOMER
AS
  CURSOR customer IS
    SELECT id,name
    FROM customers123;
    my_id customers123.id%TYPE;
    my_name customers123.name%TYPE;
```

```
BEGIN
   open customer;
   LOOP
      fetch customer into my_id,my_name;
       EXIT WHEN customer%NOTFOUND;
       DBMS_OUTPUT.PUT_LINE(my_name);
   end loop;
end;
```

PostgreSQL -

```
CREATE OR REPLACE  PROCEDURE GET_CUSTOMER()
AS
$$
DECLARE
  customer CURSOR IS
    SELECT id,name
    FROM customers123;
    my_id DECIMAL(38,0);
    my_name VARCHAR(20);

BEGIN
    open customer;
    LOOP
       fetch customer into my_id,my_name;
        EXIT WHEN NOT FOUND;
        RAISE NOTICE '%',(my_name);
    end loop;
end;
$$ LANGUAGE PLPGSQL;
```

Actual Object Count	Estimated Effort
3	0 minutes

Help Url

https://www.postgresql.org/docs/11/sql-createprocedure.html

SEQUENCE

Issue Detail

Sequences are fully supported, but all call to sequence_name.NEXTVAL or sequence_name.CURRVAL will be transformed into NEXTVAL('sequence_name') or CURRVAL('sequence_name').

Recommendation

Sequences are fully supported, but all call to sequence_name.NEXTVAL will be transformed into NEXTVAL('sequence_name')

Oracle - Sequence creation syntax:

CREATE SEQUENCE demo_seq MINVALUE 1 START WITH 1 INCREMENT BY 1 CACHE 20;

PostgrreSQL- Sequence creation syntax:

CREATE SEQUENCE demo_seq START 1;

Oracle: demo_seq.NEXTVAL;

PostgreSQL:

SELECT nextval('demo_seq'); ----- result -1

For exclusively setting the sequence value use the below syntax

SELECT setval('serial', 5);

SELECT nextval('demo_seq'); ----- result - 6

Actual Object Count	Estimated Effort
2	8 minutes

Help Url

https://www.postgresql.org/docs/9.5/sql-createsequence.html

SYNONYM

Issue Detail

SYNONYMs will be exported as views. SYNONYMs do not exists with PostgreSQL but a common workaround is to use views or set the PostgreSQL search_path in your session to access object outside the current schema.

Recommendation

SYNONYMs will be exported as views.

SYNONYMs do not exists with PostgreSQL but a common workaround is to use views or set the PostgreSQL search_path in session to access object outside the current schema.

Synonym for sequence can not be converted as view but will be converted in following way.

let's consider a conversion example of an Oracle synonym for the following very simple sequence.

An example of the Oracle sequence

CREATE SEQUENCE sample_sequence

START WITH 100

INCREMENT BY 1

NOCACHE

NOCYCLE;

Synonym for above sequence :

CREATE PUBLIC SYNONYM sample_sequence_synonym FOR sample_sequence;

So, we will use the following PostgreSQL function to emulate Oracle synonym to the above-mentioned sequence.

CREATE SEQUENCE SAMPLE_SEQUENCE INCREMENT BY 1 START WITH 100;

PostgreSQL wrapping function to emulate Oracle synonym

CREATE OR REPLACE FUNCTION sample_sequence_synonym()

RETURNS BIGINT LANGUAGE SQL AS

$$ SELECT NEXTVAL('sample_sequence'); $$;

Actual Object Count	Estimated Effort
23	19 minutes

Help Url

https://stackoverflow.com/questions/45238572/synonym-support-on-postgresql?noredirect=1&lq=1

TABLE

Recommendation

CREATE TABLE will create a new, initially empty table in the current database. The table will be owned by the user issuing the command.

If a schema name is given (for example, CREATE TABLE myschema.mytable ...) then the table is created in the specified schema. Otherwise it is created in the current schema.

CREATE TABLE table_name(

column1 datatype,

column2 datatype,

column3 datatype,

columnN datatype,

PRIMARY KEY(one or more columns)

);

Some of the important parameters of Table:

TEMPORARY or TEMP - If specified, the table is created as a temporary table.

UNLOGGED - If specified, the table is created as an unlogged table. Data written to unlogged tables is not written to the write-ahead log , which makes them considerably faster than ordinary tables.

COLLATE - The COLLATE clause assigns a collation to the column (which must be of a collatable data type).

If not specified, the column data type's default collation is used.

INHERITS (parent_table [, ...]) -The optional INHERITS clause specifies a list of tables from which the new table automatically inherits all columns.

DEFAULT default_expr - The DEFAULT clause assigns a default data value for the column whose column definition it appears within.

CHECK (expression) - The CHECK clause specifies an expression producing a Boolean result which new or updated rows must satisfy for an insert or update operation to succeed.

Example of Check constraint and default constraint :

```
CREATE TABLE EmpSalary (
    Empid     integer ,
    Salary    numeric CHECK (salary > 0),
    IsActive  boolean DEFAULT true
);
```

Actual Object Count	Estimated Effort
90	1224 minutes

Help Url

No Record Found

TABLE PARTITION

Issue Detail

Partitions are exported using table inheritance and check constraint. Hash and Key partitions are not supported by PostgreSQL and will not be exported.

Recommendation

Partitioning refers to splitting what is logically one large table into smaller physical pieces.

The table partitioning mechanism in PostgreSQL differs from Oracle. Partitioning in PostgreSQL is implemented using "table inheritance".

Based on value of partition key all rows are inserted into partitioned table. Each partition contains a subgroup of data as per the defined range of partitions.

Range and list are the supported portioning methods in PostgreSQL and these are defined on the basis of range of key values and list of key values.

Range Partitioning :

The table is partitioned into “ranges” defined by a key column or set of columns, with no overlap between the ranges of values assigned to different partitions.

For example, one might partition by date ranges, or by ranges of identifiers for particular business objects.

List Partitioning :

The table is partitioned by explicitly listing which key values appear in each partition.

Steps of Implementing List /Range “Table Partitioning”

- Create the “master” table, from which all of the partitions will inherit. This table will contain no data.
- Create several “child” tables that each inherit from the master table. Normally, these tables will not add any columns to the set inherited from the master.
- Add non-overlapping table constraints to the partition tables to define the allowed key values in each partition.
- For each partition, create an index on the key column(s), as well as any other indexes you might want.
- To redirect the data into the APPROPRIATE PARTITION, WE WILL CREATE suitable trigger function to the master table.
- Create a database trigger to redirect data inserted into the parent table to the appropriate child table.
- Ensure that the constraint exclusion configuration parameter is not disabled in PostgreSQL.conf. If it is, queries will not be optimized as desired.

Actual Object Count	Estimated Effort
7	8 minutes

Help Url

No Record Found

VIEW

Issue Detail

Views are fully supported but can use specific functions.

Impacted Object	Impacted Type	Estimated Effort
revenue	View	36 minutes

Recommendation

Views are fully supported, but if you have updatable views you will need to use INSTEAD OF triggers.

Some of the view specific parameter which are supported in oracle but not compatible with PostgreSQL are listed below.

Oracle View Parameter	Description	PostgreSQL Compatible
CREATE OR REPLACE	Re-create an existing view (if one exists) or create a new view.	Yes
FORCE	Create the view regardless the existence of the source tables or views and regardless to view privileges.	No
VISIBLE | INVISIBLE	Specify if a column based on the view will be visible or invisible.	No
WITH READ ONLY	Disable DML commands.	No
WITH CHECK OPTION	Specifies the level of enforcement when performing DML commands on the view	Yes

Example for applying With CHECK OPTION on the view :

```
CREATE TABLE DEPARTMENTS(DEPARTMENT_ID INT, DEPARTMENT_NAME
VARCHAR(100), MANAGER_ID INT, LOCATION_ID INT)

INSERT INTO DEPARTMENTS values(1,'Sales',1,1600)
INSERT INTO DEPARTMENTS values(2,'Sales',1,1700);
INSERT INTO DEPARTMENTS values(3,'Sales',1,1700);
INSERT INTO DEPARTMENTS values(4,'Sales',1,1700);
INSERT INTO DEPARTMENTS values(5,'Sales',1,1500);

CREATE OR REPLACE VIEW VW_DEP AS
SELECT DEPARTMENT_ID, DEPARTMENT_NAME, MANAGER_ID, LOCATION_ID
```

```
FROM

DEPARTMENTS

WHERE

LOCATION_ID=1700 WITH LOCAL CHECK OPTION;
```

Impact of the with check option on above view. when Location_ID is updated through view with any value other than 1700,it is not permitted.

```
UPDATE VW_DEP SET LOCATION_ID=1600;
```

SQL Error: ERROR: new row violates check option for view "vw_dep"

Actual Object Count	Estimated Effort
1	36 minutes

Help Url

https://ora2pg.darold.net/documentation.html#Migration-cost-assessment