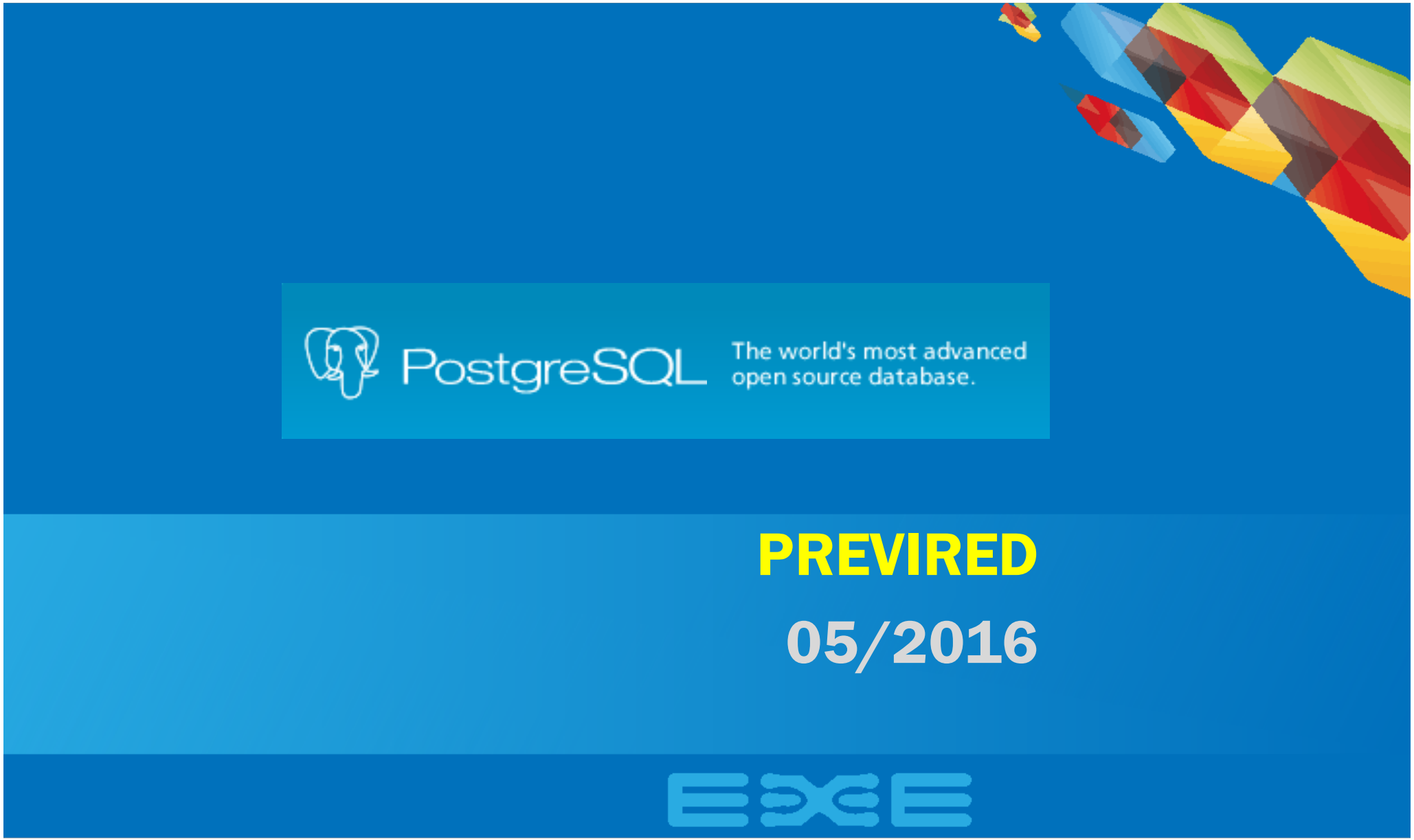
PostgreSQL
The world's most advanced
open source database.
PREVIRED
05/2016

Tipos de datos

Introducción

- **PostgreSQL soporta creación de tablas estándar SQL**
- **Pero además soporta:**
  - –Temporary (Temporales)
  - –Unlogged (Sin Log)
  - –Inherited (Con herencia)
  - –Typed (A partir de tipos)
  - –Foreign (Foráneas)
- **Un Ejemplo:**

```
CREATE TABLE logs ( log_id serial PRIMARY KEY,
user_name varchar(50),
description text,
log_ts timestamp with time zone NOT NULL DEFAULT current_timestamp);

CREATE INDEX idx_logs_log_ts ON logs USING btree (log_ts);
```

CREATE TABLE

```
CREATE [ [ GLOBAL | LOCAL ] { TEMPORARY | TEMP } | UNLOGGED ]
TABLE
[ IF NOT EXISTS ] table_name ( [  { column_name data_type [ COLLATE
collation ] [ column_constraint [ ... ] ]    | table_constraint    |
LIKE source_table [ like_option ... ] }    [, ... ]] )[ INHERITS (
parent_table [, ... ] ) ][ WITH ( storage_parameter [= value] [, ... ] )
| WITH OIDS | WITHOUT OIDS ][ ON COMMIT { PRESERVE ROWS | DELETE ROWS |
DROP } ][ TABLESPACE tablespace_name ]
```

Temporales

- Las tablas temporales son creadas con el calificador TEMPORARY o TEMP
- Las tablas temporales son eliminadas (Dropped) automáticamente al final la sesión o al finalizar la transacción (ON COMMIT)
- Las tablas permanentes con el mismo nombre no son visibles durante la sesión. Aunque las permantes pueden ser referenciadas utilizando el esquema.
- Cualquier indice creado también es descargado al final de la sesión.
- El daemon autovacuum no tiene acceso a estas tblas por lo que operaciones de vacuum o de análisuis deben ser hechas vía comandos SQL en la misma sesión.
  - Por ejemplo, si la tabla temporal participa de queries complejos es necesario ejecutar ANALYZE antes realizarlos y después de haber poblado de datos la tabla.
- GLOBAL y LOCAL están obsoletos y no hacen diferencia.

Herencia

- **PostgreSQL soporta herencia de tablas**
  - Al parecer es el único RDBMS que soporta herencia (AFAIK)
- **La tabla hija "hereda" todas las columnas de la tabla "padre"**
  - La tabla hija es creada con sus columnas y con todas las columnas de la tabla padre
  - La relación padre-hija es almacenada por PostgreSQL
  - Cualquier cambio posterior en la tabla padre es reflejado en la tabla hija
  - Al consultar por la tabla padre, las filas de la tabla hija se incluyen
  - No todo se hereda. Lo que no se hereda
    - Primary keys
    - Unique Constraints
    - Indexes
  - Check constraints se heredan, pero se pueden agregar otras

```
CREATE TABLE logs_2011 (PRIMARY KEY(log_id)) INHERITS (logs);
CREATE INDEX idx_logs_2011_log_ts ON logs USING btree(log_ts);
ALTER TABLE logs_2011 ADD CONSTRAINT chk_y2011
CHECK (log_ts >= '2011-1-1'::timestamptz
AND log_ts < '2012-1-1'::timestamptz );
```

Tablas sin log

- Para datos que pueden ser reconstituidos en caso de falla del disco o que no es necesario restaurarlos se puede utilizar la opción UNLOGGEDE
- Estas tablas no son parte de los log write-ahead. Si accidentalmente se desenfucha el servidor al momento de encenderlo y reiniciar , todos los datos se perderan durante el proceso de rollback.
- pg_dump permite no respaldar los datos unlogged
- data.
- Ejemplo

```
CREATE UNLOGGED TABLE web_sessions ( session_id text PRIMARY KEY, add_ts
time
stamptz, upd_ts timestamptz, session_state xml);
```

- La gran ventaja de estas tablas es que son muy rápidas. 15 veces más rápidas en la práctica.
- Si el servidor falla, PostgreSQL realizará un truncate de todas las tablas unlogged. (Si, Truncate signigica que borra todas las filas)
- Las tablas Unlogged no soportan los índices GiST

Tablas a partir de tipos

- Cada vez que se crea un tabl, PostgreSQL automáticamente crear tipo de dato compuesto que corresponde a tabla.
- A partir de la versión 9.0 es posible utilizar un tipo compuesto como plantillla para la creación de tablas.

```
CREATE TYPE usuario AS (nombre varchar(50), pwd varchar(10));
```

- Luego es factible crear talas con filas que son instancias de este tipo utilizando la claúsula OF

```
CREATE TABLE super_usuarios OF usuario (CONSTRAINT pk_su PRIMARY KEY
(usuario));
```

- Cuando se crean tablas a partir de tipos no es factible modificar las columnas vía ALTER.
- Sin embargo, los cambios hechos al tipo se propagan automáticamente.
- Los cambios deben ser hechos con CASCADE

```
ALTER TYPE usuario ADD ATTRIBUTE telefono varchar(10) CASCADE;
```

Constraints

- Las restricsiones en PostgreSQL constraints son las más avanzadas y quizás las más complejas que cualquier otra base de datos (AFAIX)
- Se pueden controlar todos los aspectos asociados a los datos, la manera de realizar cascada, condiciones, índices, etc.
- Para efectos de este curso revisaremos:
  - Foreign key
  - Unique
  - Check
  - Exclusion

Foreign keys

- PostgreSQL sigue las mismas reglas que todas las bases de datos para integridad referencial.
- Es factible aplicar reglas de cascada de actualización y borrado para evitar registros huérfanos.
- RESTRICT es el comportamiento por omisión

```
set search_path=censo, public;
ALTER TABLE hechos ADD CONSTRAINT fk_hechos FOREIGN KEY (tipo_hecho_id)
REFERENCES tipo_hecho(tipo_echo_id)
ON UPDATE CASCADE ON DELETE RESTRICT;
CREATE INDEX fki_facts_1 ON facts (fact_type_id);
```

Unique Constraints

- Cada tabla solo tiene una llave primaria.
- Para indicar que el valor de una columna es único se utilizan las unique constraints o los unique indexes.
- Al agregar una unique constraint automáticamente se crea un unique index asociado
  - Esto es similar a las llaves primarias
- Las unique key constraints pueden participar en la part REFERENCES de una foreign key y no pueden tener valores NULL.
- Un unique index sin una unique key constraint permite valores NULL

```
ALTER TABLE logs_2011 ADD CONSTRAINT uq UNIQUE (user_name,log_ts);
CREATE UNIQUE INDEX name ON table (column [, ...]);
```

- Para agregar restricciones de unicidad a un subconjunto de datos se pueden utilizar unique index parciales.

Check Constraints

- Las Check constraints son condiciones que deben ser satisfechas por un campo o por un grupo de campos de una fila.
- El planificador de queries toma ventajas de las check constraints y descarta aquellos que no cumplen las restricciones.

```
ALTER TABLE logs ADD CONSTRAINT chk CHECK (user_name =
lower(user_name));
```

- Otro ejemplo:

```
CREATE TABLE productos(numero integer,numero text,
precio numeric CONSTRAINT precio_positivo CHECK (precio> 0));
```

- asdasd

Exclusion Constraints

- Aparecieron en la versión 9.0.
- Estas restricciones permiten incorporar operadores adicionales para reforzar la unicidad y que no pueden ser satisfechas por una doncidicón de igualdad.
- Son muy útiles para resolver problemas de agendamiento.
- En PostgreSQL 9.2 aparecieron los tipos de datos de rango , que son muy usados para las exclusiones.
- Las exclusiones funcionan muy bien enconjunto con lo índices GiST indexes, pero también es factible usar índices compuestos B-Tree
  - –Se debe instalar la extensión btree_gist extension.
- Ejemplode uso de exclusioens es el agendamiento de un recurso. Supongamos que existen una cierta cantidad de salas de reuniones y queremos prevenir que dos reuniones calcen en horario en la misma sala.
  - –Se puede usar el operador && para chequear el traslape de horarios y la igualda para el número de sala
- CREATE TABLE reuniones(id serial primary key, sala smallint, horario tstzrange);
- ALTER TABLE reuniones ADD CONSTRAINT ex_reuniones
- EXCLUDE USING gist (sala WITH =, horario WITH &&);
- PostgreSQL automácticamente crea un índice asociado a la restricción