SQL – Creating Tables and Views

Dr. Villanes

Creating a New Table

Creating a New Table

There are three ways to create new tables in SQL:

Method 1	Copy columns and rows from existing tables
Method 2	Copy columns but no rows from an existing table.
Method 3	Define only the columns in the SQL code.

```
+--------+--------------+------------+--------+---------+
| emp_id | emp_name     | hire_date  | salary | dept_id |
+--------+--------------+------------+--------+---------+
|      1 | Ethan Hunt   | 2001-05-01 |   5000 |       4 |
|      2 | Tony Montana | 2002-07-15 |   6500 |       1 |
|      3 | Sarah Connor | 2005-10-18 |   8000 |       5 |
|      4 | Rick Deckard | 2007-01-03 |   7200 |       3 |
|      5 | Martin Blank | 2008-06-24 |   5600 |    NULL |
+--------+--------------+------------+--------+---------+
```

Method 1: Copy columns and rows from existing table(s)

CREATE TABLE ***table-name*** **AS**
SELECT …;

PostgreSQL

```
create table "Jupiter".temp as
select "Employee_ID"
from "Jupiter".employee_phones
except
select "Employee_ID"
from "Jupiter".employee_addresses
order by "Employee_ID";
```

SQLite

```
create table temp as
select movie_name as name
from movies
where movie_name like 'A%'
UNION
select genre as name
from genres
where genre like 'B%';
```

Method 2: Copy columns but no rows from an existing table

```
CREATE TABLE table2
LIKE table1;
```

```
CREATE TABLE table2 AS
SELECT * FROM mytable1
WHERE 0;
```

PostgreSQL

```
CREATE TABLE "Jupiter".new_staff
(like "Jupiter".salesstaff including
all)
```

SQLite

```
CREATE TABLE copied AS
SELECT * FROM records where 0;
```

Method 3: Define only the columns in the SQL code.

```
CREATE TABLE table_name (
   column1 datatype,
   column2 datatype,
   column3 datatype,
  ....
);
```

PostgreSQL

```
CREATE TABLE "Jupiter".films (
    code        char(5) PRIMARY KEY,
    title       varchar(40) NOT NULL,
    did         integer NOT NULL,
    date_prod   date,
    kind        varchar(10)
);
```

SQLite

```
CREATE TABLE contacts (
 contact_id integer PRIMARY KEY,
 first_name text NOT NULL,
 last_name text NOT NULL,
 email text NOT NULL UNIQUE,
 phone text NOT NULL UNIQUE
);
```

Loading Data

Adding Data to a Table

The INSERT statement can be used to **add data to an empty table, or to append data** to a table that already contains data.

Method	Description	Syntax
1	One clause per row using positional values	**INSERT INTO** *table-name* <*(column list)*> **VALUES** (*value,value,...*)**;**
2	A query returning multiple rows based on positional values	**INSERT INTO** *table-name* <(*column list*)> **SELECT** *columns* **FROM** *table-name***;**

Method 1: One clause per row using positional values

INSERT INTO table-name <(column list)>
VALUES (value,value,...);

PostgreSQL

```
Insert into "Jupiter".temp
 values ('30353') , ('48424')
```

SQLite

```
INSERT INTO people
(id, name)
VALUES
(3000,'Jack Smith')

DELETE FROM people WHERE id=3000;
```

The order of the columns in the column list is independent of the order of the columns in the table.

Method 2: A query returning multiple rows based on positional values

INSERT INTO *table-name* <(*column list*)>

SELECT *columns* **FROM** *table-name***;**

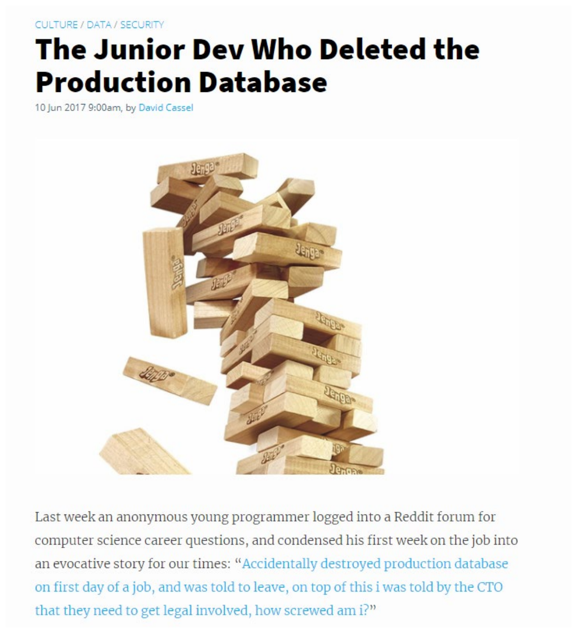
PostgreSQL

```
insert into "Jupiter".temp
select "Employee_ID" from
"Jupiter".employee_addresses
```

SQLite

```
INSERT INTO exercise.countries2 SELECT
id,name FROM exercise.countries;
```

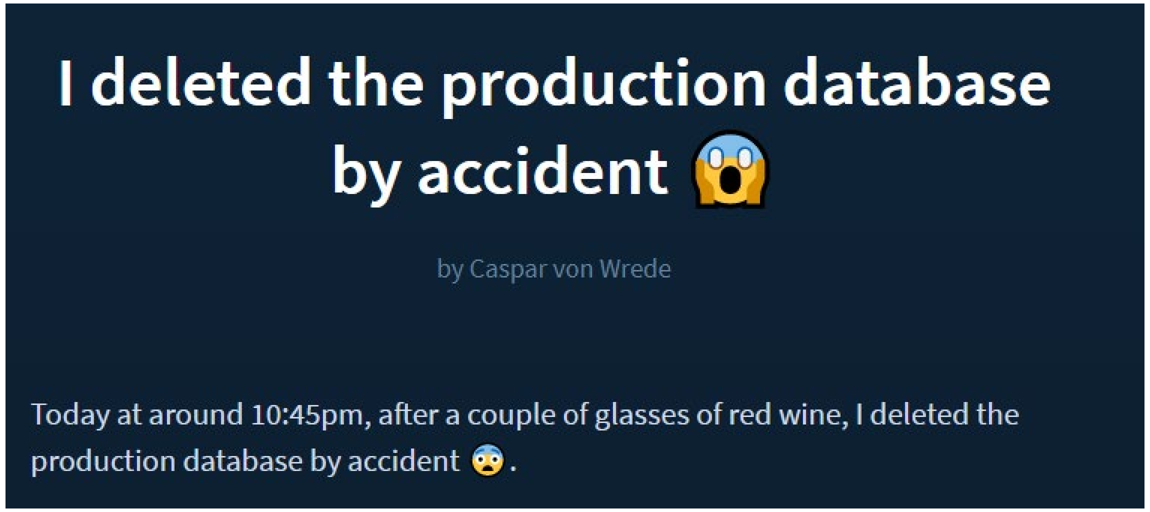
Deleting


CULTURE / DATA / SECURITY

The Junior Dev Who Deleted the Production Database

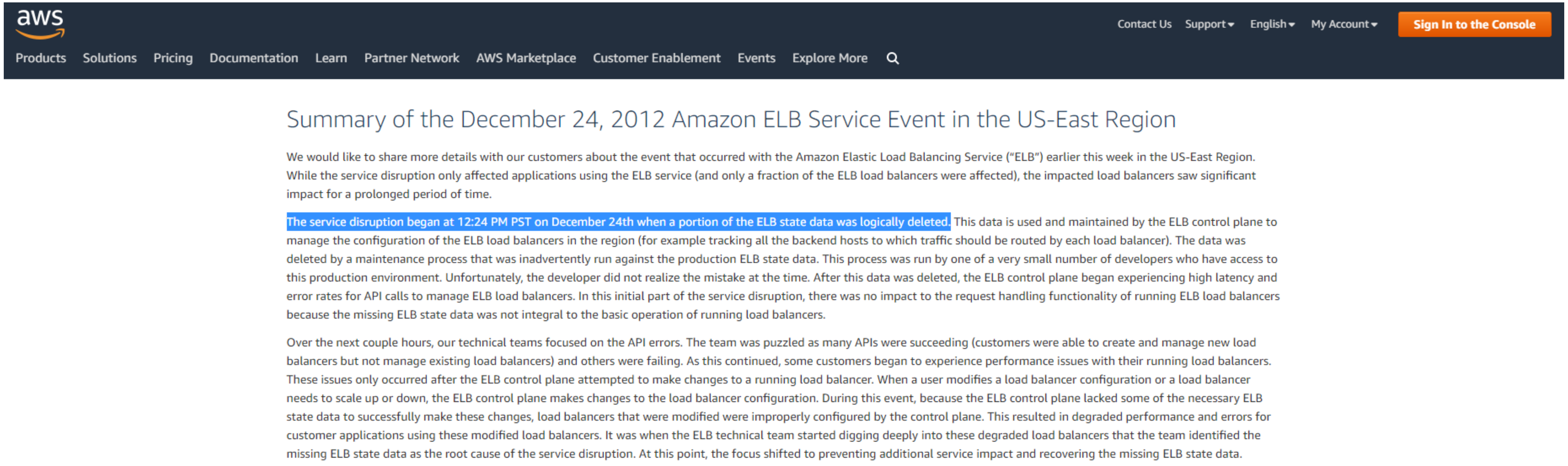
10 Jun 2017 9:00am, by David Cassel

Last week an anonymous young programmer logged into a Reddit forum for computer science career questions, and condensed his first week on the job into an evocative story for our times: “Accidentally destroyed production database on first day of a job, and was told to leave, on top of this i was told by the CTO that they need to get legal involved, how screwed am i?”


I deleted the production database by accident 😱

by Caspar von Wrede

Today at around 10:45pm, after a couple of glasses of red wine, I deleted the production database by accident 😨.



Summary of the December 24, 2012 Amazon ELB Service Event in the US-East Region

We would like to share more details with our customers about the event that occurred with the Amazon Elastic Load Balancing Service (“ELB”) earlier this week in the US-East Region. While the service disruption only affected applications using the ELB service (and only a fraction of the ELB load balancers were affected), the impacted load balancers saw significant impact for a prolonged period of time.

The service disruption began at 12:24 PM PST on December 24th when a portion of the ELB state data was logically deleted. This data is used and maintained by the ELB control plane to manage the configuration of the ELB load balancers in the region (for example tracking all the backend hosts to which traffic should be routed by each load balancer). The data was deleted by a maintenance process that was inadvertently run against the production ELB state data. This process was run by one of a very small number of developers who have access to this production environment. Unfortunately, the developer did not realize the mistake at the time. After this data was deleted, the ELB control plane began experiencing high latency and error rates for API calls to manage ELB load balancers. In this initial part of the service disruption, there was no impact to the request handling functionality of running ELB load balancers because the missing ELB state data was not integral to the basic operation of running load balancers.

Over the next couple hours, our technical teams focused on the API errors. The team was puzzled as many APIs were succeeding (customers were able to create and manage new load balancers but not manage existing load balancers) and others were failing. As this continued, some customers began to experience performance issues with their running load balancers. These issues only occurred after the ELB control plane attempted to make changes to a running load balancer. When a user modifies a load balancer configuration or a load balancer needs to scale up or down, the ELB control plane makes changes to the load balancer configuration. During this event, because the ELB control plane lacked some of the necessary ELB state data to successfully make these changes, load balancers that were modified were improperly configured by the control plane. This resulted in degraded performance and errors for customer applications using these modified load balancers. It was when the ELB technical team started digging deeply into these degraded load balancers that the team identified the missing ELB state data as the root cause of the service disruption. At this point, the focus shifted to preventing additional service impact and recovering the missing ELB state data.

DELETE Statement

DELETE FROM *table-name*
WHERE *condition***;**

If you omit a WHERE clause, then the DELETE statement deletes all the rows from the specified table

Views

What is a View?

- **Virtual table** based on the result-set of an SQL statement: **stored query**
- Contains rows and columns, just **like a real table**
- Contains **no actual data**
- **Extracts underlying data** each time it is used and accesses the most current data
- Can be **referenced** in queries in the same way as a data table

CREATE VIEW [OR REPLACE] *view-name* **AS SELECT …;**

Creating a View

CREATE VIEW *view-name*
AS SELECT …;

PostgreSQL

```
create view jupiter.Tom_Zhou as
   select Employee_Name,
          Job_Title,
          Salary
from jupiter.employee_addresses as a,
           jupiter.employee_payroll as p,
           jupiter.employee_organization as o
      where a.Employee_ID=p.Employee_ID and
            o.Employee_ID=p.Employee_ID and
            Manager_ID=120102;
```

SQLite

```
CREATE TEMP VIEW v_movies
AS
select p.name, count(*) as count_movies
from people p, people_movies pm
where p.id=pm.person_id
group by 1;
```

Views: **Advantages**

You can use views to do the following:

- **avoid** storing copies of large tables.
- **avoid** a frequent refresh of table copies. When the underlying data changes, a view surfaces the most current data.
- **pull** together data from multiple database tables and multiple libraries or databases.
- **simplify** complex queries.
- **prevent** other users from inadvertently altering the query code.

Views: **Disadvantages**

- Because views access the most current data in changing tables, the **results might be different each time you access the view**.
- Views can require significant resources each time that they execute. With a view, you **save disk storage space at the cost of extra CPU and memory usage**.
- When accessing the same data several times in a program, use a table instead of a view. This ensures consistent results from one step to the next and can significantly reduce the resources that are required.