What is lurking in your backups?

Ben Lenard[1], Alexander Rasin[1], Nick Scope[1], and James Wagner[2]

[1] DePaul University, Chicago, IL 60604, USA
[2] The University of New Orleans, New Orleans, LA. 70148, USA

Abstract. Best practices in data management and privacy mandate that old data must be irreversibly destroyed. However, due to performance optimization reasons, old (deleted or updated) data is not immediately purged from active database storage. Database backups that typically work by backing up table and index pages (rather than logical rows) greatly exacerbate the privacy problem of the old surviving data. Copying such deleted data into backups ensures that unknown quantities of old data can be stored indefinitely.
In this paper, we quantify the amount of deleted data retained in backups by four major representative databases, comparing the default behavior versus an explicit defrag operation. We review the defrag options available in these databases and discuss the impact they have on eliminating old data from backups. We demonstrate that each database has a defrag mechanism that can eliminate most of old deleted data (although in Oracle pre-update content may survive defrag). Finally, we outline the factors that organizations should consider when deciding whether to apply defrag prior to executing their backups.

Keywords: Databases · Privacy · Data Retention · Backups · Compliance

1 Introduction

Previous research has shown that database management systems (DBMS) retain copies of deleted data within internal storage [28,16]. These unallocated records and values are kept in database pages until they are overwritten by new data or by defragmenting the data structure (e.g., rebuilding the table). The extent of data that remains accessible can only be quantified using forensic tools [29] that reconstruct database contents from a full storage snapshot (including unallocated disk space or OS paging files). This deleted-but-accessible data is a potential liability that violates the best practices for data privacy. For example, government agencies, such as United States Department of Defense (DoD), publish general requirements and guidelines for best practices for storing and destroying data [9]. Other groups, such as the International Data Sanitization Consortium release guidelines for destroying data that is no longer needed to preserve privacy [15]. Moreover, some recent legislation (see Section 2) aims to grant more control to data owners. Although logically (but not physically) deleted data presents a privacy risk, it can be measured [28] and gradually rectifies itself as old data is naturally overwritten; this process can be manually expedited by additional database activity or defragmenting storage.

Interestingly, creation of backups greatly exacerbates this problem of unaccounted deleted data in storage. Both of these activities are normal – backups are required for disaster recovery and old data remains in storage because of how DBMSes manage and optimize their internal storage. However, the the interaction between these two processes creates a new and unique problem. The backup of deleted data may make it impossible for data stewards to meet the guidelines for data privacy. As we discuss in Section 2, the majority of backups rely on block backup operations. That is, the pages (rather than individual records) from tables and indexes in storage are copied into the backup. As a result, some of the deleted (but accessible) data is permanently preserved in the backup. This causes two major issues: 1) such data can never be destroyed while the backup still exists (deleted data is never going to be overwritten or defragmented in a backup) and 2) it becomes significantly more difficult to identify what deleted data remains accessible in the multitude of backups (as backups should be done regularly and are retained for a long time). Our contributions are:

- We evaluate and quantify deleted data retained in backups in four chosen representative DBMSes: Oracle 19c, Db2 11.5.4 (LUW), PostgreSQL 12, and MariaDB 5.5.
- For each DBMS, we evaluate methods to minimize the amount of deleted data introduced into backups. These methods use well-documented, existing commands, which are traditionally used for routine maintenance operations.
- We outline what factors organizations should consider when determining if and what defrag operations should be applied before backups.

2 Background and Related Work

Backups: DBMS backups are an integral part of business continuity practices to support disaster recovery. There are many mechanisms for backing up a database [12] both at the file system level and internal to the DBMS. File system backups range from a full backup with an offline, or quiesced, database to a partial backup at file system level that incrementally backs up changed files. Most DBMS platforms provide backup utilities for both full and partial backups.

Even for a relatively simple task of coping database files at the file system level, there are various methods ranging from a simple copy command with an offline database to a more complex copy of the storage system in an active database [23]. Moreover, one can replicate database changes from one database to another to provide a live back up or replica by using tools such as Oracle Data Guard [22], in addition to taking traditional backups of the database.

The type of backup depends on two application metrics: Recovery Point Objective (RPO) and Recovery Time Objective (RTO); as your RPO and RTO shorten, the complex solutions such as Oracle Data Guard [22] emerge. RTO is defined as the time it takes to recover the database after a crash and RPO is defined as how close to the crash or error can you restore the database. A backup solution could range from a simple DBMS backup utility to snapshotting a filesystem once the database is quiesced, to replicating the database and then backing up the database, or other options [22]. The criticality of the application and data as well as RTO and RPO determines the backup solution and its complexity; in other words, how much would downtime or the financial and reputation loss of data could cost your organization. In addition to data, one might also backup transaction logs, archievelogs (Oracle), logarchive (Db2), binary log (MySQL), or write ahead log (WAL in Postgres), file backups, to replay transactions to meet the RPO goal of the application. For example, one would restore last backup before the failure and then replay the transaction logs to meet the RPO set by the organization's needs. For the purposes of this paper we focus on the backups provided by the relational DBMSes since backups are used most often and replication mechanisms do not protect against logical corruption (unless caught before the log file is applied).

For the commercial products we tested, Oracle provides RMAN, Recovery Manager, and IBM Db2 provides its own backup utility. Oracle RMAN and Db2 backup utilities provide for an online backup of the database that incorporates activity that took place during the backup. These utilities provide block-level backups with either a full backup of the database or a partial backup with database pages that changed since the last backup. For the purposes of this paper, partial backups can be *incremental* or *delta*. For example, if we took a full backup on Sunday and daily partial backups and needed to recover on Thursday, database utilities would restore the full backup from Sunday and then either 1) apply delta backups from Monday, Tuesday, and Wednesday or 2) apply Wednesday's incremental backup.

PostgreSQL's Pg_dump utility [25] generates a consistent backup of the database at the time of the invocation in a file containing SQL statements (a logical representation of the database state). PostgreSQL also offers `pg_basedump` which is a binary of the database to which the WAL files can be applied; in other words, you can backup the WAL files and replay during a restore operation. The version of MariaDB located in the CentOS 7 repository provides the mysqldump utility which is also a logical dump of the database.

Filesystem level snapshots as a method of backup are created by quiescing the database and then invoking a snapshot command ether on the storage system, such as Dell EMC [23] or Netapp [20], or on the filesystem itself such as ZFS [24]. Such approach requires significantly more storage space, but can be used when running the database backup would be disruptive to the application.

Database storage: A database stores data in pages that contain the object data as well as meta data about the row itself [28]. When a database deletes a record, the record is logically deleted in the page similar to a file delete in a filesystem [27]. This is done for performance optimization reasons. Since typical backup utilities use block backup, rather then a logical representation like `pg_dump` or `mysqldump`, these deleted records also propagate to backup files. Block backup is normally used because it is impractical to convert a large database into a logical dump.

We know that a small amount of the database changes in enterprise applications, between 5% and 18% [14], therefore a logical dump of a database is inefficient.

Perhaps surprising to some, tape backups are still widely used in the industry, including by cloud providers. For example, in 2011 when a Google Gmail upgrade failed, affecting 150,000 accounts, Google used off-site tapes to restore impacted accounts [17]. Another paper illustrated the regular use of tape backups in 2017 with offsite providers [19] for medical systems. While tapes are a cost-efficient medium for backups, they make it nearly impossible to identify or purge deleted data that was accidentally backed up with active database content.

Retention policies: Data management requirements define how long data should be stored or when the data must be destroyed. These retention policies can come from various sources such as government legislation, internally from an organization, or the result of various lawsuits. For example, recently passed in the European Union, the General Data Protection Regulation [5] greatly expanded consumer power over personal data. On 25 May 2018 (the same day GDPR took effect), Google was accused of not complying with GDPR [13], which eventually led to a fine of €50 million. In 2019, Deutsche Wohnen was accused of using systems which were unable to delete data resulting in €14.5 million fine [11]. Thus, companies not complying with data retention requirements could be subject to large financial penalties.

SSBM and database workload: Database community used TPC-H [10] and SSBM [10] for a wide variety of research projects and experiments. SSBM [21] is a star schema benchmark that has simplified the TPC-H schema to represent a typical simple data warehouse. Hsu et al. [14] enumerated typical distributions within real-world workloads of a transaction processing database. Depending on the industry, Hsu et al. observed anywhere from 96% reads to 80% reads in different workloads; we rely on their estimates to create our own workload using SSBM. Exact balance of a workload depends on many factors and varies by industry, but our goal is to approximate a generic realistic workload (rather than to model a particular industry or scenario).

SysGen: Our experiments leverage publicly available SysGen [18] developed by Lenard et al. and an SSBM-based workload generator included with SysGen. The workload generator is a Python script configured with the number of iterations, or runs that consists of a sequence of select, insert, update, and delete statements. SSBM is limited to `SELECT` statements so the SysGen workload generator created inserts, updates, and deletes according to average workload balance (derived from [14]). Insert generating function increments last primary key of the `customer` table by one, and generates a random date inline with the other fields. For the updates, we update the `customer` table, with a random selection of 5 different update templates. For deletes, we delete a random customer ID and their purchases from `lineorder` table.

SysGen [18] includes a set of scripts that 1) create virtual machines, 2) configure their OSes according to our settings, 3) deploy a DBMS with our configuration, 4) execute given workloads in the DBMS, 5) capture storage snapshots (disk and RAM). For this paper, we extracted the backups from disk, whether a filesystem snapshot or the output from a utility, for each evaluated DBMS.

Terminology: We use several Oracle-specific terms in Section 4, which we define here. Oracle assigns an address to each row which is exposed through the pseudo-column `ROWID` to the `SELECT` clause [6]. `ROWID` is tied to the physical location of the row in the Oracle files and will change when the row moves. High Water Mark (HWM) is a reference that indicates the boundary of the table in the file (i.e., the largest range of space that the table occupied at any point of time). For example, if you had a table with a 1,000,000 rows, deleted all of the rows and then inserted 100,000 rows, HWM would mark the location where 1,000,000th row was originally stored. HWM is is used by Oracle to decide how far down the tablespace to scan for table data blocks [26].

3 Design and Setup

3.1 Hypervisor Setup

For our experiments we used a server with dual Xeon E5645 processors, 64GB of RAM, and an SSD for storage. We used CentOS 7 x86_64 as base operating system of the hypervisor on a VMWare Workstation Pro 16 (referred to as "host" in this paper). The host also has an NFS share exported which is discussed in this section. We provided ample RAM storage to avoid swapping on the host.

3.2 Virtual Machine Setup

Each database VM server consists of 8GB of RAM, 4 vCPUs, 1 x vNIC and a 25GB VMDK file. The VMDK file was partitioned into: 350MB /boot, 2GB swap, and the remaining storage was used for the / partition; this was done with standard partitioning and EXT4 as the filesystem. Individual VM servers used CentOS 7 as well; we refer to these as the database server in this paper. The database server OSes were configured identically except for dependent packages for the tested DBMS. The only exception to this rule is the Oracle VM where we allocated 35GB to the VMDK file since Oracle's install is roughly 6GB by itself; Oracle also requires custom kernel configuration settings which are done by their RPM on install. All VM's have an NFS mount point exported from the host machine so that we can examine the backup contents external to the VM. For Oracle and Db2, we let the DBMS automatically manage memory allocation for the instances. For Postgres and MariaDB, this option was not available so we allocated 80% of the VM's RAM to the database instance. MariaDB used the InnoDB storage engine for storing the data; other DBMSes do not provide multiple storage engine options. In all DBMSes, We enabled archievelogging (Oracle), logarchive(Db2), the binarylog (MariaDB), and WAL (Postgres).

In addition to the database server VMs, we run a client VM which executes the workload against the database server remotely; the client has 2 vCPU's, 2GB of RAM, 1 x vNIC, CentOS 7 x86_64 and a similar partitioning scheme to the database server VMs. Each client VM has the RDBMS' client software installed as SQL Scripts, DDL, and raw table data. We separated the client from the database server to represent an N-Tiered or client server architecture. Lastly, on the servers and the clients, we installed the SysGen scripts so that automated execution of the backup tests can be achieved. This consisted of RPMs which contain helper scripts, SSH keys and the disabling of Firewalld and SeLinux.

3.3 Dataset and Workload

To simulate a realistic database, we used the SSBM schema with two minor modifications. We loaded data into the five tables at SSBM Scale 4 which yields 120,000 customer (one of the 5 tables) records. For the purposes of this experiment, we added a second index on the Customer name which we used to track records in storage. Where possible, we separated the table data from the index data in storage to simplify storage analysis; that was feasible in every database except MariaDB because it uses index organized tables (IOT).

SysGen allows executing a series of workload passes. A pass is a workload batch that we execute before executing backup actions and taking a storage snapshot of the database VM. We executed the following number of SQL Statements per pass: 0 selects, 266 inserts, 266 updates, and 532 deletes. Since `SELECT` statements do not affect what we are measuring, we focused on inserts, updates, and deletes. There are 532 deletes since we executed 266 deletes against the `Customer` table. `Lineorder` table is linked to the `Customer` table by a foreign key (`lo_custkey`). There is only one unique customer in the `Customer` table, but more than one `Lineorder` record can be deleted for a given customer. While the percentage of change to the `Customer` table is low, 0.22%, per iteration, or 1.10% at the end of the passes, one could argue that Customers in an enterprise do not often get deleted. Furthermore, we are not defining an iteration as a unit of time but rather as a unit of work since we have seen in [14] that workloads vary by industry; the extension of that would vary by application type.

4 Experiments

4.1 Execution

We used SysGen to create and load the dataset into each database server VM. On completion of the initial load step, SysGen suspends the VM using vmware commands and copies the VM contents to another location on disk. We used VM snapshot to execute consistent experiments starting from the same initial OS and DBMS state for different backup types.

Post load, the `SysGen runtests-DBTYPE-run.sh` executed multiple workload passes against each database platform and created backups as configured. After the initial load we executed five workload passes against a given database platform, and it is the same workload executed across different platforms. At the end of the pass, we perform the database backup and then we snapshot the VM, via pausing the VM and then using Linux `cp` to copy off all of the VM files on the hypervisor's filesystem. We measured at least two runs per database.

The first run serves as a comparison baseline. The second run is where we try to remove deleted data by defragmenting the table within the RDBMS platform; we execute a defrag at the end of each pass prior to the backup.

For each run, the group of 5 iterations, baseline, defrag, and the different backup types, we always capture the backups at the end of each pass as well as the VM's contents after we suspend the VM, and copy them for future analysis.

4.2 Backup Types Tested

Oracle and Db2 supply backup utilities for binary backups of the database – these backups could be full or partial. For the full backup, everything contained in the backup is required to restore the database to the time of the backup. Db2 and Oracle also provide a method to create a partial backup; partial backups contain only the changed pages of the database. Furthermore, these partial backups can be broken down further into cumulative and delta backups as discussed in Section 2. One would chose a delta or incremental backup based on how much storage they want to allocate for backups as well as their RTO.

Open-source versions of Postgres and MariaDB do not provide binary backup tools, and their dump commands is akin to running a `SELECT` statement which would not provide any useful information; therefore we used filesystem snapshots as the backup method. While we shutdown MariaDB and Postgres to perform a consistent backup of the filesystem, one might do this on a live database, assuming the storage system provided snapshot capabilities. MariaDB and Postres, like Db2 and Oracle, support commands that force a live database into a consistent state for the purposes of a backup. For example, Postgres has `pg_start_backup()` and `pg_stop_backup()` which instruct the database to become consistent so that a live database's files can be copied. We use the Postgres start backup command, then use the storage system to take a snapshot, and then let Postgres resume operations. Oracle and Db2 also support the functionality to take a storage system snapshot. While the speed of the snapshot is subject to a multitude of factors, it is usually a relatively quick operation; run time depends on the amount of data that has changed. While copying files is not exactly the same as cloning a filesystem in storage, it does provide a similar result. MariaDB has `FLUSH TABLES tbl_list WITH READ LOCK` for similar functionality.

One might ask why we did not take storage snapshots in Oracle and Db2; their utilities were used since they provide additional functionality such as checking the database for corrupted pages and verifying the backup files. If an organization is going to rely on these commercial vendors, they are going to rely on the toolkits provided by the vendor to ensure their data can be restored.

The target of all backup commands was an NFS share exported from the hypervisor to avoid contamination of the DB Server VM as well as ease of analysis.

4.3 Removing Deleted Data

One of the primary purposes of a defragment operation is to reduce the size of the object and to purge unused data. While SQL-92 is standardized across different platforms, the concept of defragment is not universal. In this section we discuss the different approaches used in the RDBMS platforms in this paper.

- **MariaDB** implements `OPTIMIZE TABLE` which creates a new copy of the table for InnoDB tables.
- **PostgreSQL** has the `VACUUM` utility that will reorganize the table by discarding obsolete data in-place [8].
- **Db2** provides a plethora of options for it's defrag command [3]; for the purposes of consistency, in this paper we used Db2 Classic Reorg which recreates the table as well as rebuilds the indexes associated with the table.
- **Oracle** does not provide a direct defrag command as other databases; the closest command to defrag is `ALTER TABLE MOVE` which moves the table between tablespaces. This operation is similar to recreating the table but one would still need to rebuild the index. Oracle also allows enabling row movement on a table which supports the functionality to shrink a table (compacting individual pages). We explored both options in this paper since they are among the recommended options to execute a defrag in Oracle [4].

4.4 Oracle Defragmentation

Since the other RDBMS platforms had simple and unambiguous built-in defrag available through the command line interface, we do not discuss the nuances of these commands. However, we describe three different defrag methods used for Oracle in our experiments. All of our defrag methods start from `ALTER TABLE customer ENABLE ROW MOVEMENT` which allows rows to move in storage when an explicit move or a shrink command is executed. Updated rows will be a deleted and re-inserted [2]. Moving the row also changes its `ROWID`.

- **Defragmentation Method 1** – Using the command `ALTER TABLE customer SHRINK SPACE CASCADE` directs Oracle to rearrange the rows as well as the tables referenced by the foreign keys of customer table and moves the high water mark (see Section 2) [7]. Following that, we rebuild all the indexes of the tables that were moved using the `ALTER INDEX [index name] REBUILD`.
- **Defragmentation Method 2** – We execute `ALTER TABLE customer MOVE` which causes a table rebuild by moving customer table from one tablespace to a different tablespace. We used the move command as one of the recommended methods to rebuild the table [1]; we have to also rebuild the indexes since they become invalid when the table moves. We use `ALTER TABLE customer SHRINK SPACE CASCADE` and `ALTER INDEX [index name] REBUILD` (as in the previous approach) to rebuild referenced tables and indexes.
- **Defragmentation Method 3** – We execute `ALTER TABLE customer MOVE` to rebuild the table as in defrag method 2. We defragment the table using `ALTER TABLE customer SHRINK SPACE COMPACT`; unlike defrag method 1, this command does not move the high water mark [7]. Finally, we use `ALTER TABLE customer SHRINK SPACE CASCADE` and `ALTER INDEX [index name] REBUILD` (as in previous approaches) to rebuild referenced tables and indexes.

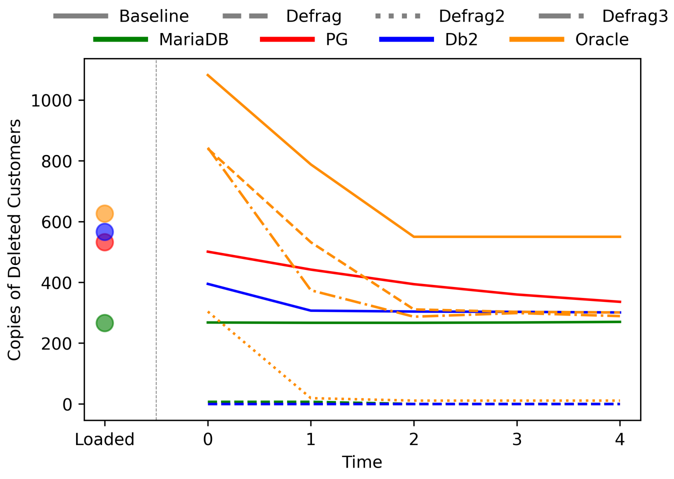

Fig. 1. Delete Counts in Full Backups

5 Results

5.1 Deleted Data in Backups

For every DBMS using a full backup, some deleted data was found in each backup in our experiments. Recall that we are only tracking the 266 original customer records deleted at Time_0 – deletes at each workload iteration will introduce *new* deleted rows that will be backed up as well. As can be seen in Figure 1, all of the DBMSes retained over 200 of the Time_0 deleted records when no defragment was executed. Defragment operation significantly or completely reduced the amount of deleted data in backups. In MariaDB, PostgreSQL, and Db2, defragmenting eliminated deleted customer records from full database backups. In Oracle, none of the three defrag methods we consider were able to completely get rid of deleted customers backups. Method 2 seemed the most effective for removing deleted data from backups.

Figure 2 summarizes the delta backup results in Oracle and Db2. Recall that our versions of MariaDB and PostgreSQL do not support incremental backups. Since defrag Method 3 in Oracle did not offer a significant advantage of Method 1, our subsequent experiments consider only Methods 1 and 2. Db2 retains approximately half of the T_0 deleted customers without an explicit defragment command, and none of the deleted customer when using defragmentation. In Oracle, defrag Method 1 eliminates most of the T_0 deleted customers; Method 2 does much more poorly, retaining more than half of T_0 deleted customer records even after 5 workloads. Since this is a delta backup, one should remember all prior backups are needed to restore the database from the last full backup. Therefore, the total number of available deleted customer records is the sum of deleted records from Figures 1 and 2.

To fully remove each batch of deleted records, all delta backups as well as all full backup must be properly defragged. Figure 3 shows the T_0 deleted customer

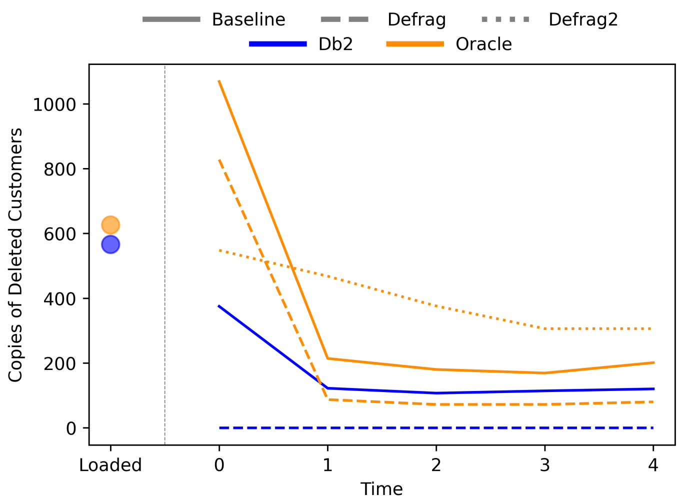

Fig. 2. Delete Counts in Delta Backups

records for incremental backups. Db2 exhibits the same pattern, backing up over half of the deleted records without defrag and none of the deleted records with backup. Oracle preserves a significant amount of T_0 deleted records, even past all 5 workloads, with some reduction caused by defrag Method 1. Defrag Method 2 eliminates the deleted customers from T_0 by the second query workload. The large amount of records in Oracle are partially explained by UNDO table space included in Oracle backup together with table data.

Similar to the delta backup method, the database requires a full backup (with the deleted records) to be restored to disk before restoring the most recent incremental backup. Unlike with delta backups, incremental backups only require the most recent backup and the full backup to be defragmented.

In order to purge deleted data, it is necessary to defrag both the full backups and any partial backups that follow (defragmenting partial backups alone cannot guarantee data destruction). While such defragmenting may be costly in terms of I/O and CPU time, it is a balancing act of the organization based on their compliance requirements. In order to make cleaning of backups more practical, organizations could choose to defrag only the tables with sensitive data before backup. In choosing the appropriate defrag level, organizations must consider their retention requirements, computational power budget, as well as database availability (to not disrupt their required system availability).

5.2 Remaining Updated Data

When updating a record, a DBMS will often perform the operation by deleting the current record and inserting the new updated row. This is always true when the new record is larger than the old record, and sometimes true when the new record is equal or smaller in size (depending on the DBMS). In this experiment, we measure the presence of old (deleted by an update) records in database backups. Similar to our deleted record experiments, we counted the number of

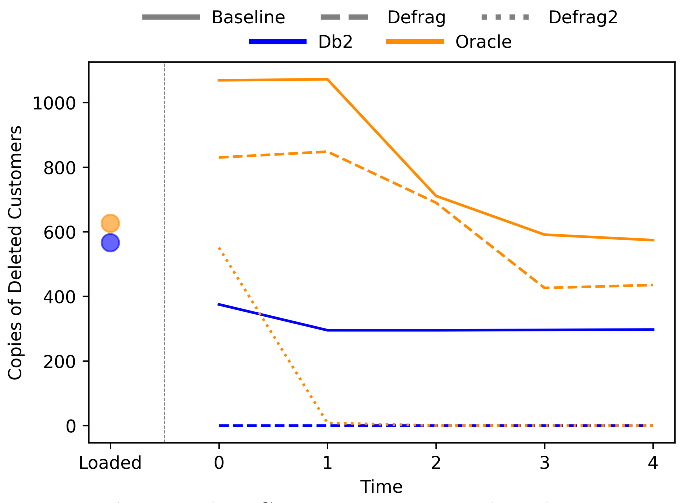

Fig. 3. Delete Counts in Incremental Backups

deleted-by-update records immediately after we loaded the database (at which point there are zero deleted records) and then again after each workload iteration.

At the load time our backup contains 266 customer records that we update in our workload during $Time_0$. After each iteration, we track the number of deleted (i.e., pre-update) customer records remaining in a backup. We only update our 266 records of interest at $Time_0$, but we insert, delete, and update other records to simulate normal database activity for subsequent workload iterations.

Figure 4 shows the counts of original pre-update 266 customer records that can be accessed in a full backup after each workload iteration. Without defrag, MariaDB contains between 1 and 4 old customer records (we believe that MariaDB backup does not generally include old customer records due to its default use of Index Organized Tables). Db2 backups include approximately 30 of the old T_0 customer records, while PostgreSQL contains a decreasing number of old records, ranging from 231 to 93 customers at $Time_4$. Oracle backup contains 223 old customer records; interestingly, these are not deleted records in table storage, but entries in UNDO tablespace that Oracle backs up with the data.

We also measured the updated records after defragmenting the table prior to backup. Figure 4 shows a noticeable drop in the number of old customers present in the backups. MariaDB contains 9 old customers in the defragmented backup, PostgreSQL contains no old customers in the backup, Db2 backups include anywhere between 1 to 5 from the original pre-update customers. Finally, Oracle shows the same behavior (approximately 230 old customers due to the inclusion of the UNDO tablespace in the backup) for defrag Methods 1 and 3. However, defrag Method 2 backup includes 4 to 5 customers (with the exception of $Time_0$, where 15 old customer records are backed up).

Index Organized Tables (IOT) in MariaDB naturally reduce the number of pre-update records stored in database backup (because records with same primary key are stored in the same location in IOT). Although Oracle supports

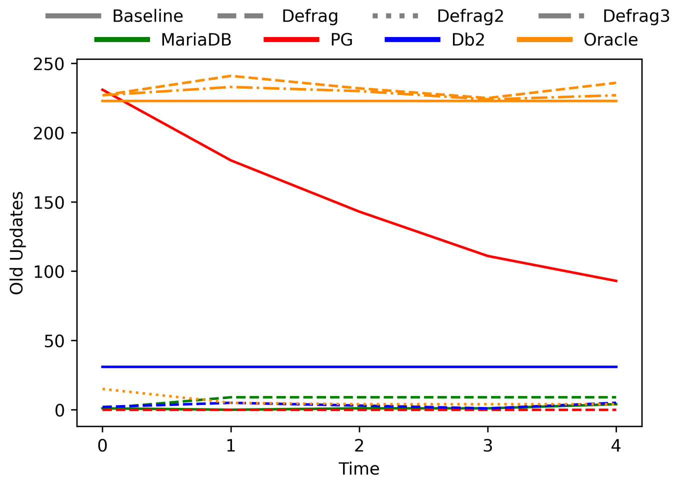

Fig. 4. Old Updates in Backups

IOTs, it would not benefit from IOT tables in the same way, because old records are copied with UNDO tablespace into the backup. PostgreSQL defrag (`VACUUM`) fully reclaims the space occupied by old records in our update experiments.

6 Conclusion

We verified that deleted records as well as old pre-update record versions live within database backups. Old data remains accessible in backups in perpetuity, until the entire backup is destroyed. Although old records are overwritten during normal database operation, each deleted record may be captured by a series of full and incremental backups before it is destroyed. While our experiments show customer records decaying from Time_0 (slowly without defrag and significantly faster with defrag), we note that it represents only the records deleted or updated at Time_0. An entirely new set of deleted records are created at Time_1 and at each subsequent workload step, which would follow the same pattern.

We demonstrated that defragmentation operations can significantly reduce the amount of deleted data captured in backups. Furthermore, the correct defragmentation option is key to expunging the old data values; while Db2, Postgres, and MariaDB have explicit commands to defragment the data structures, Oracle does not and as a result we tested three different defrag methods. One might have to measure deleted data retention in order to verify that they are choosing the right defrag method. Although defragmentation commands are costly, we argue that they are necessary to support good data privacy practices. We recommend that organizations defragment tables with sensitive data before backup and verify that their chosen defrag method is effective in eliminating deleted data. In cases where unknowingly retaining data is a significant liability for an organization, additional privacy tools will be necessary.

With data retention, organizations are at times are obligated to fully purge all traces of necessary data from their systems (lest organizations be subject to fines as with [11]). We have shown that backups have the potential to preserve unknown amounts of delete data in perpetuity (for the lifetime of backups). Ultimately, DBMS vendors may have to provide backup tools that incorporate defrag functionality, giving organizations control over their data.

7 Future Work

We intend to perform additional experiments with other database settings to determine if we can reduce the amount of deleted data left in backups without utilizing a costly (in terms of I/O operations) defrag commands. Future directions will also test against different workloads that model different industries, to evaluate when workload type influences the amount of remaining deleted data and whether the best defrag method can be chosen analytically.

Destroying deleted data may be accomplished by a custom trigger that explicitly updates values in-place before deletion. Such operation would not eliminate the deleted row itself, but it would sanitize the individual values of the deleted records. The overhead of this approach would need to be measured against the overhead of a defragmentation. If trigger-based approach is viable, database platforms could include an internal sanitization feature to eliminate deleted data.

Acknowledgments

This work was partially funded by the US National Science Foundation Grant IIP-2016548 and by Argonne National Laboratory.

References

1. Alter table move, `https://asktom.oracle.com/pls/apex/asktom.search?tag=alter-table-move`
2. Enable row movement, `https://asktom.oracle.com/pls/apex/asktom.search?tag=enable-row-movement`
3. Ibm db2 reorg, `https://www.ibm.com/support/producthub/db2/docs/content/SSEPGG_11.5.0/com.ibm.db2.luw.admin.cmd.doc/doc/r0001966.html`
4. What is the difference between shrink, move and impdp, `https://asktom.oracle.com/pls/apex/asktom.search?tag=what-is-the-difference-between-shrink-move-and-impdp`
5. Gdpr archives (2020), `https://gdpr.eu/tag/gdpr/`
6. Sql language reference (Sep 2020), `https://docs.oracle.com/en/database/oracle/oracle-database/19/sqlrf/ROWID-Pseudocolumn.html#GUID-F6E0FBD2-983C-495D-9856-5E113A17FAF1`
7. Sql language reference (Sep 2020), `https://docs.oracle.com/en/database/oracle/oracle-database/19/sqlrf/ALTER-TABLE.html#GUID-552E7373-BF93-477D-9DA3-B2C9386F2877`
8. Vacuum (Nov 2020), `https://www.postgresql.org/docs/12/sql-vacuum.html`
9. Assistant Secretary of Defense for Networks and Information Integration: Electronic records management software applications design criteria standard. `https://www.esd.whs.mil/Portals/54/Documents/DD/issuances/dodm/501502std.pdf` (2007)

10. Barata, M., Bernardino, J., Furtado, P.: An overview of decision support benchmarks: Tpc-ds, tpc-h and ssb. New Contributions in Information Systems and Technologies pp. 619–628 (2015)
11. Betschka, J., Kiesel, R., Christ, S.: Deutsche wohnen muss 14,5 millionen euro strafe bezahlen (May 2019), `https://www.tagesspiegel.de/berlin/rekordbussgeld-wegen-datenschutzverstoessen-deutsche-wohnen-muss-14-5-millionen-euro-strafe-bezahlen/25191038.html`
12. Dudjak, M., Lukić, I., Köhler, M.: Survey of database backup management. In: 27 th International Scientific and Professional Conference"Organization and Maintenance Technology (2017)
13. Fox, C.: Google hit with £44m gdpr fine over ads (Jan 2019), `https://www.bbc.com/news/technology-46944696`
14. Hsu, W.W., Smith, A.J., Young, H.C.: Characteristics of production database workloads and the tpc benchmarks. IBM Systems Journal **40**(3), 781–802 (2001)
15. International Data Sanitization Consortium: Data sanitization terminology and definitions (Sep 2017), `https://www.datasanitization.org/data-sanitization-terminology/`
16. Jeon, S., Bang, J., Byun, K., Lee, S.: A recovery method of deleted record for sqlite database. Personal and Ubiquitous Computing **16**(6), 707–715 (2012)
17. Langer, M.: Developing a data backup strategy. Risk Management **64**(10), 12–13 (2017)
18. Lenard, B., Wagner, J., Rasin, A., Grier, J.: Sysgen: system state corpus generator. In: Proceedings of the 15th International Conference on Availability, Reliability and Security. pp. 1–6 (2020)
19. Lozupone, V.: Disaster recovery plan for medical records company. International Journal of Information Management **37**(6), 622–626 (2017)
20. Netapp: (July 2020), `https://kb.netapp.com/Advice_and_Troubleshooting/Data_Protection_and_Security/Snap_Creator_Framework/What_is_Snap_Creator`
21. O.Neil, P., et al.: The star schema benchmark and augmented fact table indexing. In: Performance evaluation and benchmarking, pp. 237–252. Springer (2009)
22. Oracle: Oracle maa reference architectures, `https://www.oracle.com/webfolder/technetwork/tutorials/architecture-diagrams/high-availability-overview/high-availability-reference-architectures.html`
23. Oracle: (Jan 2018), `https://www.delltechnologies.com/en-us/collaterals/unauth/white-papers/products/storage/h14232-oracle-database-backup-recovery-vmax3.pdf`
24. Oracle: (May 2019), `https://docs.oracle.com/cd/E53394_01/html/E54801/gbciq.html#scrolltoc`
25. Shaik, B.: Backup and restore best practices. In: PostgreSQL Configuration, pp. 93–110. Springer (2020)
26. sqlandplsql: High water mark oracle (May 2020), `https://sqlandplsql.com/2013/05/22/high-water-mark-oracle/`
27. Wagner, J., Rasin, A., Grier, J.: Database forensic analysis through internal structure carving. Digital Investigation **14**, S106–S115 (2015)
28. Wagner, J., Rasin, A., Grier, J.: Database image content explorer: Carving data that does not officially exist. Digital Investigation **18**, S97–S107 (2016)
29. Wagner, J., Rasin, A., Malik, T., Heart, K., Jehle, H., Grier, J.: Database forensic analysis with dbcarver. In: CIDR 2017, 8th Biennial Conference on Innovative Data Systems Research (2017)