WAL Internals Of PostgreSQL

Contents

REDO Definition

- Redo logs contain a history of all changes made to the database.
- Redo log files are used by
  - Recovery
  - Incremental Backup and Point In Time Recovery
  - Replication
- Every change made to the database is written to the redo log file before it is written to the data file.
- The redo log buffer is flushed to the redo log file when a COMMIT is issued.
- A background log writer process to flush Redo in case if the database setting is such that Redo should be flushed in a batch.
- Redo is not required for temporary tables.

Jargons

- WAL – Write Ahead Log. It is used in context of transaction log files.
- Xlog - Transaction log. It is used in context of transaction log buffers.
- LSN – Log sequence number. It is used to mark position of log in pages.
- Bgwriter – Background writer.This is used to flush shared buffers and perform checkpoint.
- Clog - Commit log. It is used in context of transaction status buffers.
- Partial Page Write – This happens when OS is able to write partial page in disk files which can cause corruption.

Redo Implementation in PostgreSQL

- In PostgreSQL, Redo logs are known by Write Ahead Logs (WAL) and it is ensured that log entries must reach stable storage before the data-page changes they describe.

- To guarantee above,
    - Each data page (either heap or index) is marked with the LSN (log sequence number --- in practice, a WAL file location) of the latest XLOG record affecting the page.
    - Before the bufmgr can write out a dirty page, it must ensure that xlog has been flushed to disk at least up to the page's LSN.

- This low-level interaction improves performance by not waiting for WAL I/O until necessary.

- Temporary table operations doesn’t get logged.

Redo Record Data

INSERT

```
INSERT INTO score
(team, runs, wickets)
VALUES
('AUS',100,4);
```

Header Info : crc,
RM_HEAP_ID, Xid,len,
XLOG_HEAP_INSERT

Data : tupleid, infobits,
c0: 'AUS'
c1: 100
c2: 4

UPDATE

```
UPDATE score
SET
   runs = 104,
   wickets = 5
WHERE team = 'AUS';
```

Header Info : crc,
RM_HEAP_ID, Xid,len,
XLOG_HEAP_UPDATE

Data : oldtupid, newtupid,
infobits,
c0: 'AUS'
c1: 104
c2: 4

DELETE

```
DELETE FROM score
WHERE team = 'AUS';
```

Header Info : crc,
RM_HEAP_ID, Xid,len,
XLOG_HEAP_DELETE

Data : tupleid

Algorithm For WAL Action

- Pin and take Exclusive-lock on buffer containing the data page to be modified
- Start the critical section which ensures that any error occur till End of critical section should be a PANIC as buffers might contain unlogged changes.
- Apply changes to buffer.
- Mark the buffer as dirty which ensures bgwriter (checkpoint) will flush this page and marking buffer dirty before writing log record ensures less contention for content latch of buffer.
- Build a record to be inserted in transaction log buffer.
- Update the Page with LSN which will be used by bgwriter or flush operation of page to ensure that corresponding log is flushed from buffer.
- End Critical section.
- Unlock and unpin the buffer.

Important Locks Used In WAL

- **WALInsertLock**
- This lock is used to insert transaction log record contents into transaction log memory buffer. First this lock is taken then whole contents including full buffers (if full_page_writes is on) are copied into log buffers.
- Other places where this lock is used
  - During flush of log buffers to check if there are any more additions to log buffer since last it is decided till log buffer flush point.
  - To determine the Checkpoint redo location
  - During Online backup to enforce Full page writes till the backup is finished.
  - to get the current WAL insert location from built in function.

Important Locks Used In WAL

- **WALWriteLock**
- This lock is used to write transaction log buffer data to WAL file. After taking this lock all the transaction log buffer data upto pre-decided point will get flushed.
- Places where it get used are
  - Flush of transaction log which can be due to Commit, Flush of data buffers, truncate of commit log etc.
  - During Switch of Xlog.
  - During get of new Xlog buffer, if all buffers are already occupied and not flushed.
  - Get the time of the last xlog segment switch

Write Ahead Log Files

- The transaction log files are stored in $PGDATA/pg_xlog directory. They are named as 000000020000070A0000008E.
  - The first 8 digits identifies the timeline,
  - The following 8 digits identifies the (logical) xlog file and
  - The last ones represents the (physical) xlog file (Segement)
- The physical files in pg_xlog directory are not actually the xlog files;
  PostgreSQL calls it segments.
- Each Segment contains Bocks of 8K and Segment size is 16M.

Block 0	Block 1	Block 2	…	Block 255
1. Seg Hdr 2. Block Header 3. WAL Records Each WAL record has header. WAL 1, 2 ,3	1. Block Header 2. WAL Records 3. Each WAL record has header. WAL 4,**5**	1. Block Header 2. WAL Records 3. Each WAL record has header. WAL **5**,6,7,8		1. Block Header 2. WAL Records 3. Each WAL record has header. WAL m,n,…

Switch Transaction Log Segment

- **What is XLOG SWITCH?**

  It means to change the current xlog segment file to next segment file.

- **What all needs XLOG SWITCH?**

- At Archive timeout, so that current files can be archived.
- At Shutdown, so that current files can be archived.
- At start of online backup
- By built-in function pg_switch_xlog

Async Commit

- In this mode, the WAL data gets flushed to disk after predefined time by a background WAL writer process.
- Commit only updates the WAL record pointer upto which background process needs to flush.
- In worst-case three walwriter cycles will be required to flush the WAL buffered data.
- We must not set a transaction-committed hint bit on a relation page and have that record make it to disk prior to the WAL record of the commit. This can maintain transaction status consistency across crash recovery.

Protection against partial writes in disk

- **Data Page Partial Write Protection**

- To protect the data page partial write, the first WAL record affecting a given page after a checkpoint is made to contain a copy of the entire page, and we implement replay by restoring that page copy instead of redoing the update.

- **WAL Page Partial Write Protection**

- Each WAL record contains CRC and checking the validity of the WAL record's CRC will detect partial write.
- Each WAL page contains a magic number, the validity of which is checked after reading each page.

XLogRecord

- Fixed size log record header which sits in the beginning of each log record.
- Stores the information of current transaction id.
- Stores CRC to validate the log record.
- Stores total length of record
- Info bits to indicate whether backup block information is present.
- Resource manager id to indicate type of resource of log record

XLogRecData

- The resource manager data is defined by a chain of one or more XLogRecData structs.
- - Multiple structs are used when backup pages needs to be written

- when the buffer is backed up, it does not insert the data pointed to by this XLogRecData struct into the XLOG record

- Flag to indicate whether free space of backup page can be omitted.

- Actual data of record

- Buffer associated with data

BkpBlock

- Header information for backup block. This is appended to Xlog Record.
- Stores information about hole inside backup page
- Stores information for relation containing block
- Follows it is the actual backup page data

Advantages/Disadvantages Of PG Implementation

- **Advantages**

1. One of the Advanced features of PostgreSQL is it its ability to perform transactional DDL's via its Write Ahead Log design.
2. Removing holes of data page and then write to WAL will have less I/O if pages are not full.
3. WAL data written for Insert and Delete operation is lesser than systems having UNDO (Oracle).
4. During ASync Commit, writing data only in blocks ensures less usage of I/O bandwidth.
5. Keeping Log Sequence Number on each page ensures that during dirty page flush Buffer Manager doesnot need to wait for Flush of WAL until necessary.

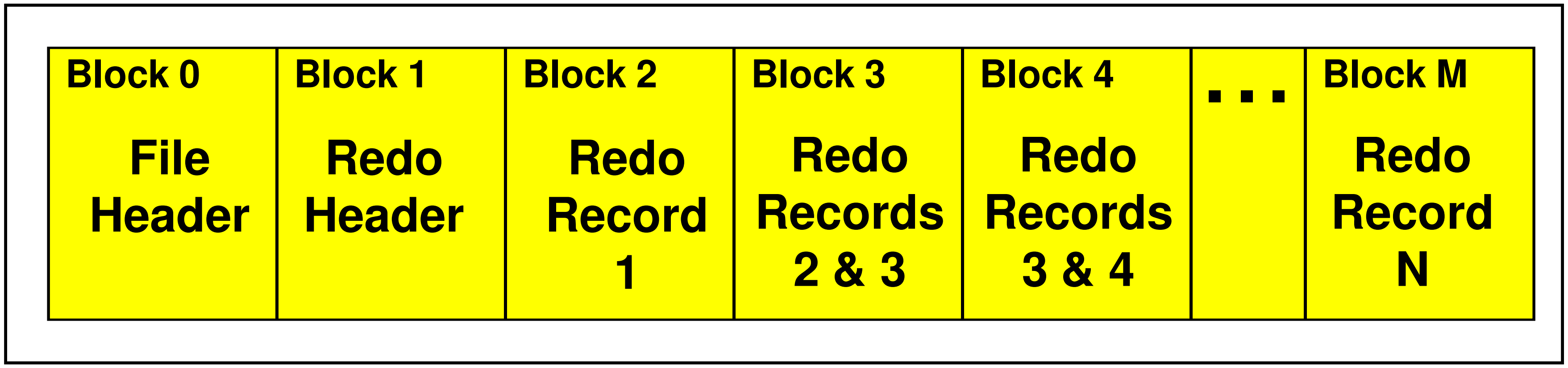
Advantages/Disadvantages Of PG Implementation

- **Disadvantages**

  1. Flushing data pages during Commit can be heavier.
  2. Update operation writes whole row in WAL even if 1 or 2 columns are modified. This can lead to increase in overall WAL traffic.
  3. During Async Commit, each time to check tuple visibility it needs to refer CLOG Buffer/File which is costly.
  4. Calculating CRC for each WAL can be costly especially in case during full data page writes.

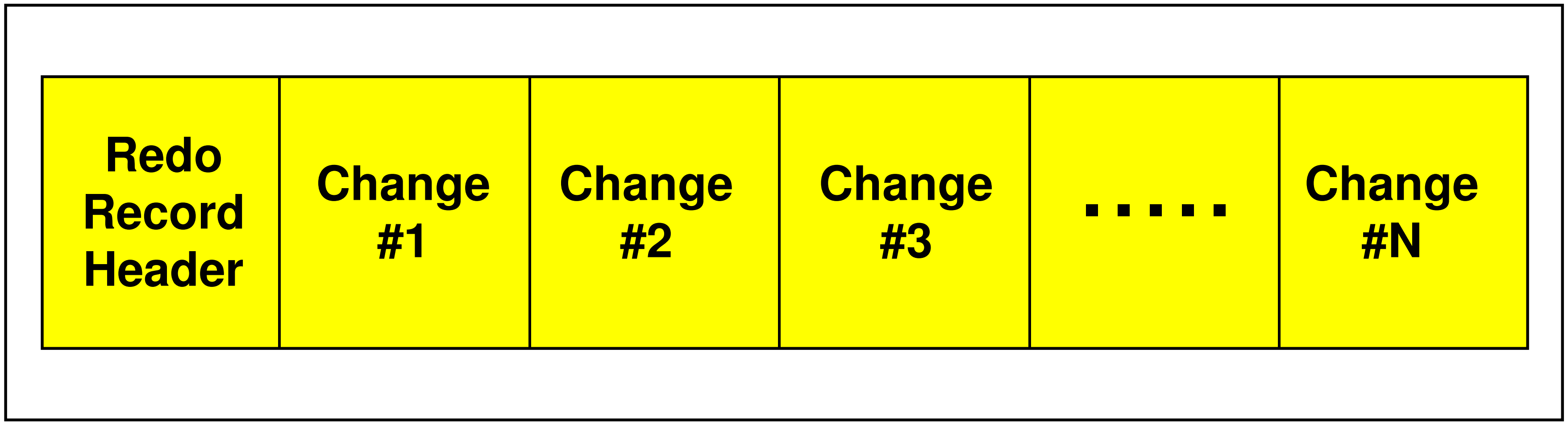
Redo Log Files

- Redo log uses operating system block size
  - usually 512 bytes
  - format dependent on
    - operating system
    - Oracle version
- Each redo log consists of
  - header
  - redo records
- Redo log is written sequentially



Redo Records

- A redo record consists of
  - redo record header
  - one or more change vectors
- Each redo record contains undo and redo for an atomic change
- Some changes do not require undo



Redo Record Header

- Every redo record has a header

```
REDO RECORD - Thread:1 RBA: 0x003666.000000cf.0010 LEN: 0x019c VLD: 0x01
SCN: 0x0000.00eb1279 SUBSCN:  1 05/08/2003 15:44:12
```

- Fields include

Thread	Thread Number
RBA	Redo Byte Address
LEN	Length of record in bytes
SCN	System Change Number
	Date and Time of Change

Redo Byte Address (RBA)

- Every redo record has a Redo Byte Address (RBA) e.g.

```
RBA: 0x003666.000000cf.0010
```

- RBA is 10 bytes in length

- RBA identifies start of redo record
- Fields are
  – Log sequence number (0x3666)
  – Block number within redo log (0xcf)
  – Byte number within block (0x10)

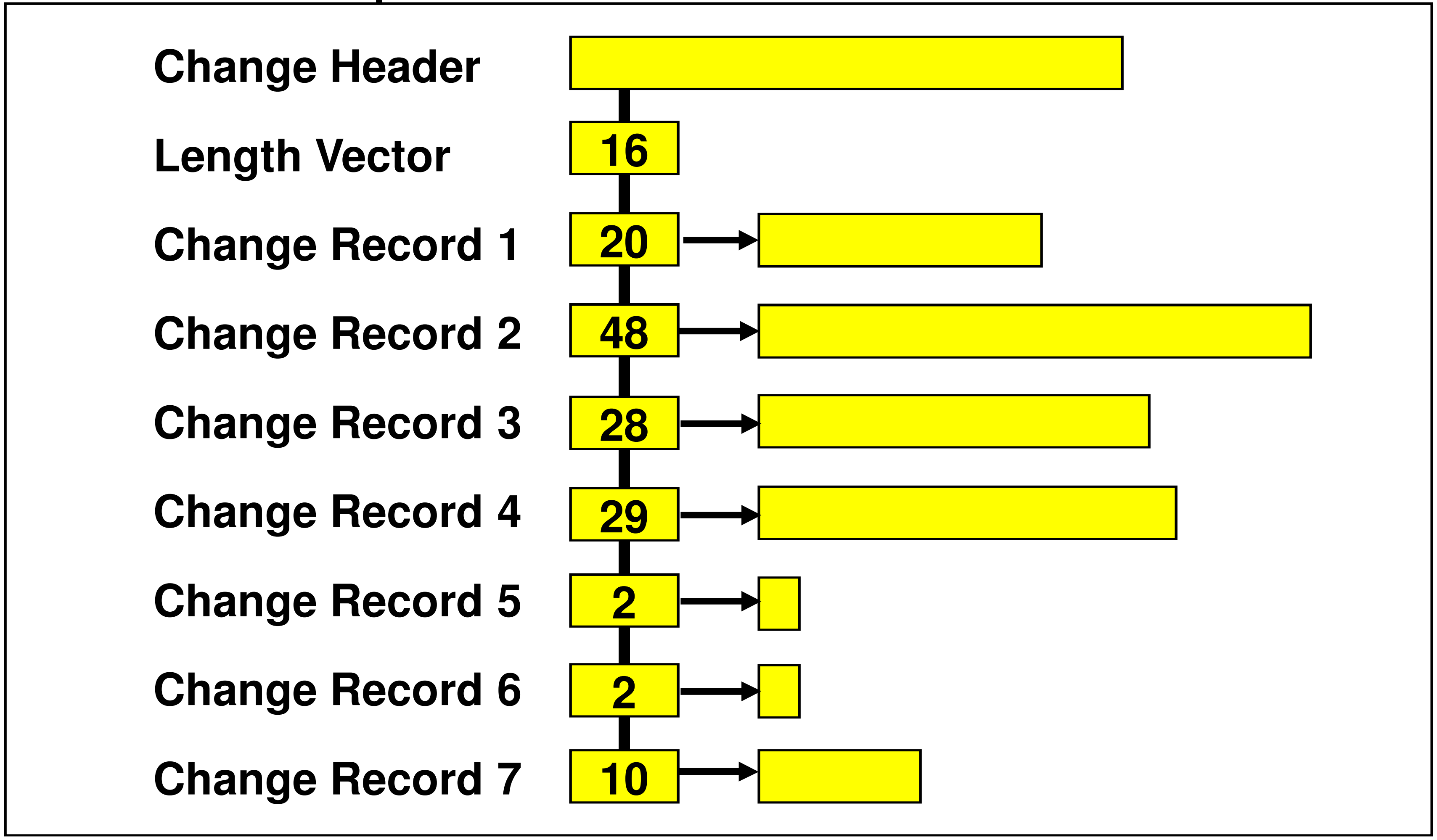
Change Vector

- Describes a change to a single data block
- Can apply to
  - undo headers
  - undo blocks
  - data segment headers
  - data blocks

- Is created in PGA before the data block buffer is modified
- Consists of
  - header
  - array of change record lengths
  - array of change records

Change Vector

- For example



Change Vector Header

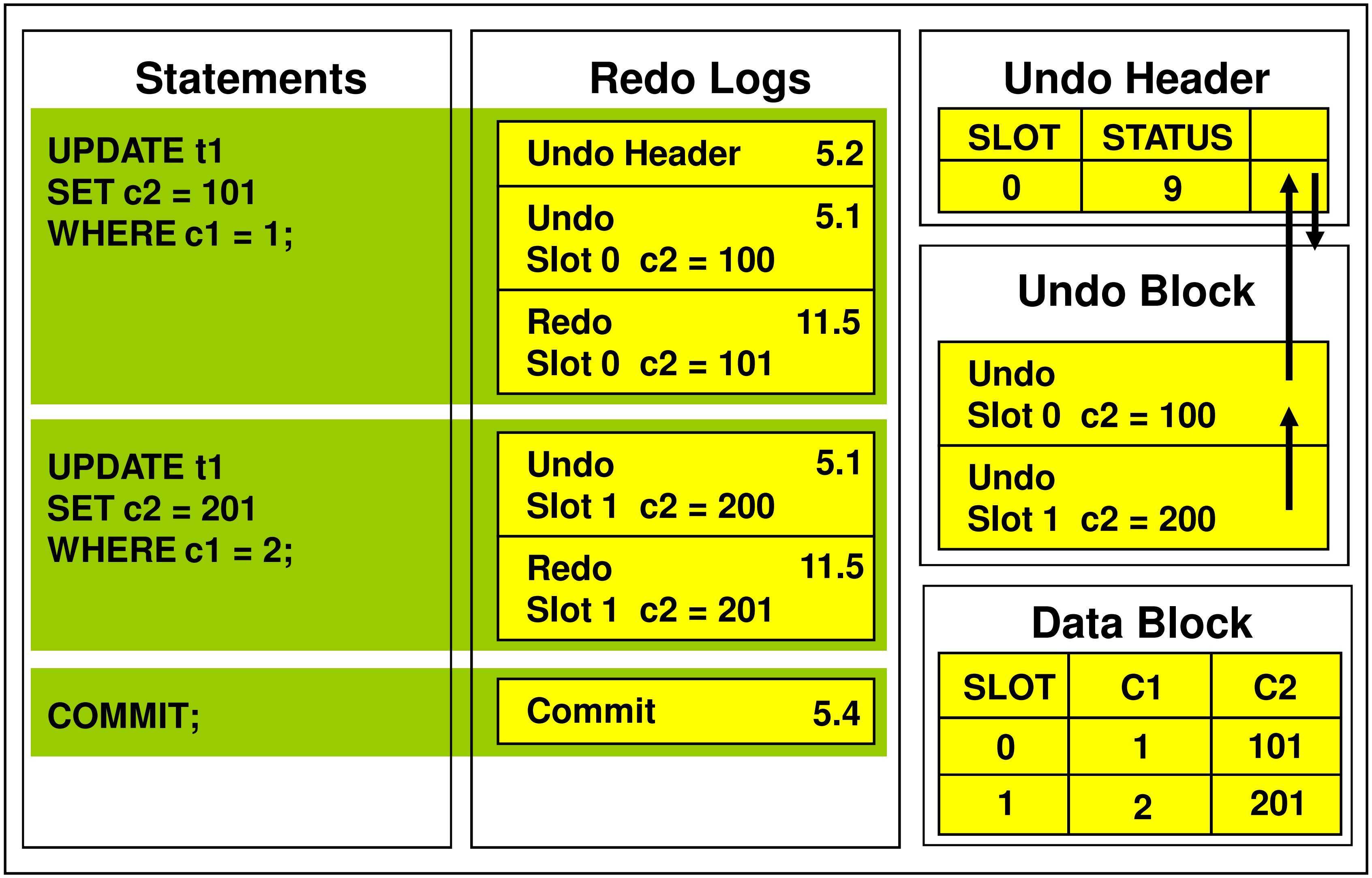
- Every change vector has a header e.g.

```
CHANGE #2 TYP:0 CLS: 1 AFN:5 DBA:0x0144d023 SCN:0x0000.0ac67cce
SEQ:  4 OP:11.5
```

- Fields include

CHANGE	Change number
TYP	Change type
CLS	Class
AFN	Absolute File Number
DBA	Relative Database Block Address
SCN	System Change Number
SEQ	Sequence Number (relative to SCN)
OP	Operation Code

Transactions



```
UPDATE t1
SET c2 = 101
WHERE c1 = 1;
```

Undo Header	5.2
Undo Slot 0 c2 = 100	5.1
Redo Slot 0 c2 = 101	11.5

```
UPDATE t1
SET c2 = 201
WHERE c1 = 2;
```

Undo Slot 1 c2 = 200	5.1
Redo Slot 1 c2 = 201	11.5

```
COMMIT;
```

Commit	5.4

Undo Header

SLOT	STATUS	
0	9	

Undo Block

Undo Slot 0 c2 = 100
Undo Slot 1 c2 = 200

Data Block

SLOT	C1	C2
0	1	101
1	2	201

Redo Record Data

- For data blocks

	INSERT	UPDATE	DELETE

	INSERT INTO score
	(team, runs, wickets)
	VALUES
	('AUS',100,4);
		UPDATE score
		SET
		runs = 104,
		wickets = 5
		WHERE team = 'AUS';
			DELETE FROM score
			WHERE team = 'AUS';

Undo	OP 5.1 (11.1) Delete Row Piece - DRP Slot 4:	OP 5.1 (11.1) Update Row Piece - URP Slot 4: c1: 100 c2: 4	OP 5.1 (11.1) Insert Row Piece - IRP Slot 4: c0: 'AUS' c1: 104 c2: 5
Redo	OP 11.2 Insert Row Piece - IRP Slot 4: c0: 'AUS' c1: 100 c2: 4	OP 11.5 Update Row Piece - URP Slot 4: c1: 104 c2: 5	OP 11.3 Delete Row Piece - DRP Slot 4:

STOP

Global Temporary Tables

Statements	Redo	
-- Statement #1 INSERT INTO t1 VALUES (1);	HEADER	5.2
	UNDO #1	5.1
	REDO #1	11.2
-- Statement #2 INSERT INTO t1 VALUES (2);	UNDO #2	5.1
	REDO #2	11.2
-- Statement #3 INSERT INTO t1 VALUES (3);	UNDO #3	5.1
	REDO #3	11.2
COMMIT;	COMMIT	5.4

Protection against partial writes in disk

- Redo Page Partial Write Protection
- We can configure Oracle to use checksums to verify blocks in the redo log files.
- If block checking is enabled, Oracle computes a checksum for each redo log block written to the current log.
- Oracle writes the checksum in the header of the block. Oracle uses the checksum to detect corruption in a redo log block.

LGWR Process

- The log writer process writes one contiguous portion of the buffer to disk. LGWR write:
  - A commit record when a user process commits a transaction
  - Redo log buffers
    - Every three seconds
    - When the redo log buffer is one-third full
    - When a DBW*n* process writes modified buffers to disk, if necessary.

LGWR Process

- In times of high activity, LGWR can write to the redo log file using group commits.
- For example, when LGWR is writing T-1 Commit operation redo entries to disk, other users issue COMMIT statements.
- After the first transaction's entries are written to the redo log file, the entire list of redo entries of waiting transactions (not yet committed) can be written to disk in one operation, requiring less I/O than do transaction entries handled individually.
- During flush of Redo log buffer if redo log buffer is partially filled then the empty space will be wasted in Redo Log File.

REDO Definition

Redo Implementation in PostgreSQL

Key Structures Used in PostgreSQL

Advantages & Disadvantages of PostgreSQL Implementation

Redo Implementation in Oracle

Advantages & Disadvantages of Oracle Implementation

Improvements in PostgreSQL

Detailed method for one of the improvements

Advantages/Disadvantages Of Oracle Implementation

- **Advantages**

1. Update has less redo as it writes only changed data.
2. Group commits by LGWR can reduce the overall I/O and improve performance.
3. Writing in block sizes same as hardware/OS block size gives benefit.
4. Log writer flushes redo if redo log buffer is 1/3rd full which will make sure there can never be contention for Redo Log Buffer.

Advantages/Disadvantages Of Oracle Implementation

- **Disadvantages**

  1. There can be lot of space wastage in Redo log files during high activity in database.
  2. Redo of Insert and Delete SQL statements will be more as compare to PostgreSQL because it has to write Redo for Undo data generated as well.
  3. Headers size is more compare to PostgreSQL. It can have multiple headers for one Redo Record. 1st for each Redo Record then for each Change Vector.

REDO Definition

Redo Implementation in PostgreSQL

Key Structures Used in PostgreSQL

Advantages & Disadvantages of PostgreSQL Implementation

Redo Implementation in Oracle

Advantages & Disadvantages of Oracle Implementation

Improvements in PostgreSQL

Detailed method for one of the improvements

Improvements in PostgreSQL

1. For Update operation the amount of WAL required can be reduced by writing only changed column values and reconstruct full row during recovery.
2. Flushing Data page contents during Commit by main user process is costly, other databases does it in background process.
3. We can introduce a concept similar to Group Commits by WAL writer which can improve performance during high volume of transactions.
4. Improve the Tuple visibility logic for setting the transaction status in a tuple during Asynchronous Commits.
5. To improve the writing of same Redo Block again and again if the transaction data is small.

Need for WAL reduction for Update

- In most telecom scenario's, the Update operation updates few columns out of all used in schema
- For example

CREATE TABLE callinfo (logtime date not null, updatetime date, callerno varchar2(20), agentid varchar2(10), status int, i0 int, c0 varchar2(20), i1 int, c1 varchar2(20), i2 int,
c2 varchar2(20), i3 int,c3 varchar2(20),i4 int,c4 varchar2(20),
i5 int,c5 varchar2(20),i6 int,c6 varchar2(20),i7 int,
c7 varchar2(20), i8 int,c8 varchar2(20),i9 int,
c9 varchar2(20),content varchar2(512));

update callinfo set status = status + 1, i0 = i0 + 1 where callerno = :callerno

Method-1 to reduce WAL for Update op

- Only send the changed data to WAL and reconstruct tuple during recovery.
- Reconstruction would need the old tuple data and the new tuple changed data to reconstruct the row at time of recovery.
- After the row is generated it will be inserted in data page.
- It is better to do apply this method when old and new tuple are on same page, otherwise it need to do I/O during recovery.
- The changed columns are logged in a byte-by-byte instruction set format using tuple descriptor.

Method-1 Contd..

- Byte-byByte format is used instead of attributes and number of attribute as we don’t tuple descriptor during recovery.

- The diff instructions allow us to express operations very simply.
  For Example,

```
CREATE TABLE foo (col1 integer, col2 integer, col3 varchar(50),
 col4 varchar(50));

INSERT INTO foo values (1, 1, repeat('abc',15), repeat(‘def’,15));
```

Method-1 Contd..

UPDATE foo SET col2 = 100 WHERE col1 = 1;

will generate diff instructions (assuming 4 byte alignment for now)

COPY 4 (bytes from old to new tuple)
IGNORE 4 (bytes on old tuple)
ADD 4 (bytes from new tuple)
COPY 90 (bytes from old to new tuple)

Method-1 Contd..

- With a terse instruction set the diff format can encode the diff instructions in a small number of bytes, considerably reducing the WAL volume.

- The simplicity of the diff algorithm is important because this introduces
  - additional CPU and
  - potentially contention also, since the diff is calculated while the block is locked.

- As a result, it is proposed that the diff would only be calculated when the new tuple length is in excess of a hard-coded limit

Method-2 to reduce WAL for Update op

- This method of reducing WAL will be applied only if table has fixed length columns(int,char,float).
- Keep only changed data and offset of it in WAL.
- Reconstruction would need the old tuple data and the new tuple changed data to reconstruct the row at time of recovery.
- After the row is generated it will be inserted in data page.
- It is better to do apply this method when old and new tuple are on same page, otherwise it need to do I/O during recovery.

Method-2 Contd..

- log the offset, length, value format for changed data to reconstruct the row during recovery.
- During recovery with this information the new row can be constructed without even tuple descriptor.
- As the log format is only for fixed length columns, so during recovery it can be directly applied at mentioned locations to generate a new tuple.
- This method can also be optimized such that it will log in described format if all changed columns are before any variable data type column.

Method-2 Contd..

- For Example

  CREATE TABLE foo (col1 integer, col2 integer, col3 varchar(50),
  col4 varchar(50));

  INSERT INTO foo values (1, 1, repeat('abc',15), repeat(‘def’,15));

  UPDATE foo SET col2 = 100 WHERE col1 = 1;

- will generate log without considering tuple header

  old tuple location

  Offset: 5, length: 4 value: 100

- offset and length can be stored in 2-3 bytes considering this will be applied tuples of length less than 2000 bytes.

Comparison for method-1 & method-2

- Method-1 is valid irrespective of data type of columns, whereas Method-2 is applied only in certain cases depending on datatype.

- In case of Method-2, contention chances will be less as the information required for logging should be available during tuple formation.

- Generation of new tuple during recovery can be faster in Method-2 as it needs to make a copy of original tuple and then replace new values at specified location.

Thank You