INFO3504 – Database Systems II (Adv)

Week 2 (Adv): Physical Row Storage Formats

Dr. Uwe Röhm
School of Information Technologies

Agenda

- Looking at PostgreSQL Page and Tuple layout
- Comparing this with other systems
- Handling of special cases:
  - NULLs
  - BLOBs

Row Storage Format Comparison

- Physical Row Formats
  - strive to be compact and flexible for variable length rows
    - Caveats: handling of NULLs
  - Row meta-data must be handled too
    - Some sytems support multiple versions of rows (cf. discussion of snapshot isolation later in week 9)
  - technical issues: some hardware architectures require values to be aligned to, e.g, word addresses

- Comparison between
  - PostgreSQL 9.0.3
  - Oracle 10g R2
  - SQL Server 2005

Row Format in PostgreSQL

http://www.postgresql.org/docs/9.0/static/catalog-pg-attribute.html

Row = RowHeader + NullBitmap + *alignment padding* + *(OID)* + RowData

- RowHeader:
  - 23 Bytes; cf. next slide
- NullBitmap
  - variable len: either 0 (if all NOT NULL) or ((|*Columns*| + 7) / 8) Bytes
  - one bit per attribute; 1 if NULL, 0 otherwise
- *OID (PostgreSQL speciality as it supports objects)*
  - *fix 4 Bytes (optional,depending whether table WITH OIDs or not)*
- RowData = FixedColumns + VarColumns
  - FixedColumns: directly stored & aligned!
  - VarColumns = varattrib + userdata + aligned
    - varattrib = 4 bytes length words including 2 bits for compression/TOAST flags

Row Format in PostgreSQL (cont'd)

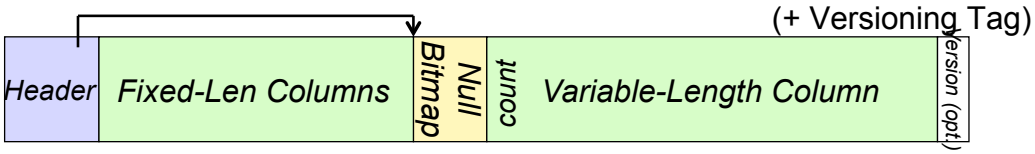
- Row Header structure
  - 23 bytes (plus bitmap plus padding; cf. *t_hoff* value as 'pointer')
  - Cf. **src/include/access/htup.h**:
    `typedef struct HeapTupleHeaderData`
  - Some information on visibility of a tuple for current transaction snapshot or newer version (needed for snapshot isolation algorithm)
    - **t_xmin** TransactionId 4 bytes insert XID stamp
    - **t_xmax** TransactionId 4 bytes delete XID stamp
    - **t_cid** CommandId 4 bytes insert CID stamp *(actual a UNION struct)*
    - **t_ctid** ItemPointerData 6 bytes current TID of this or newer row version
  - How long is this row? Is it variable length? Does it have NULLs?
    - **t_natts** int16 2 bytes number of attributes
    - **t_infomask** uint16 2 bytes various flag bits
      - e.g. HAS_NULL | HASVARWIDTH | HASOID | locks(!)
    - **t_hoff** uint8 1 byte /* sizeof header incl. bitmap, padding */

PostgreSQL Docu Excerpt

Cf. PostgreSQL 9 Documentation, section 54.5:
(http://www.postgresql.org/docs/9.0/static/storage-page-layout.html):
[...]
"All table rows are structured in the same way. There is a fixed-size header (occupying 23 bytes on most machines), followed by an optional null bitmap, an optional object ID field, and the user data. The header is detailed in Table 54-4. The actual user data (columns of the row) begins at the offset indicated by t_hoff, which must always be a multiple of the MAXALIGN distance for the platform. The null bitmap is only present if the HEAP_HASNULL bit is set in t_infomask. If it is present it begins just after the fixed header and occupies enough bytes to have one bit per data column (that is, t_natts bits altogether). In this list of bits, a 1 bit indicates not-null, a 0 bit is a null. When the bitmap is not present, all columns are assumed not-null. The object ID is only present if the HEAP_HASOID bit is set in t_infomask. If present, it appears just before the t_hoff boundary. Any padding needed to make t_hoff a MAXALIGN multiple will appear between the null bitmap and the object ID. (This in turn ensures that the object ID is suitably aligned.)"
[…]
"interpreting the actual data can only be done with information obtained from other tables, mostly **pg_attribute**. The key values needed to identify field locations are attlen and attalign. There is no way to directly get a particular attribute, except when there are only fixed width fields and no null values. All this trickery is wrapped up in the functions **heap_getattr**, **fastgetattr** and heap_getsysattr."
[…]
"To read the data you need to examine each attribute in turn. First check whether the field is NULL according to the null bitmap. If it is, go to the next. Then make sure you have the right alignment. If the field is a fixed width field, then all the bytes are simply placed. If it's a variable length field (attlen = -1) then it's a bit more complicated. All variable-length datatypes share the common header structure varattrib, which includes the total length of the stored value and some flag bits. Depending on the flags, the data may be either inline or in a TOAST table; it might be compressed, too (see Section 54.2)."

Row Format: SQL Server 2005

Row = RowHeader + FixedData + NullBitmap + VariableData

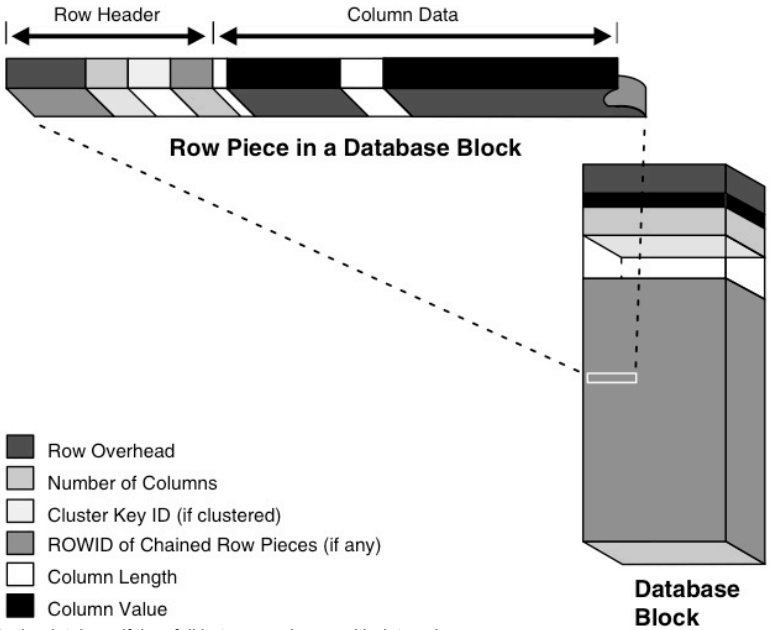


- Row Header = 4 bytes
  - two bytes of record metadata (record type)
  - two bytes pointing forward in the record to the NULL bitmap
- Fixed-Length Data
  - all fixed-length columns stored together just after header
- NullBitmap
  - Length: 2 + ((|*Columns*| + 7) / 8) Bytes
    - two bytes for count of columns in the record
    - variable number of bytes to store one bit per column in the record, regardless of whether the column is nullable or not
      (SQL Server 2000 had one bit per nullable column only -> complex)

Row Format: SQL Server 2005 (cont'd)

- Variable-Length Column Data
  - Variable_Data_Size = **2** + (*Num_Variable_Cols* x 2) + *Max_Var_Size*
    - **two bytes for the count of variable-length columns**
    - two bytes per variable length column, giving the offset to the start of the column value
- Versioning tag (optional)
  - this is in SQL Server 2005+ only
  - a 14-byte structure that contains a timestamp plus a pointer into the version store in tempdb

Row Format in Oracle 10g



"Nulls are stored in the database if they fall between columns with data values. In these cases they require 1 byte to store the length of the column (zero).
…
To conserve space, a null in a clumn only stores the column length (zero). Oracle does not store data for the null column. Also, for trailing null columns, Oracle does not even store the column length."

[Oracle 10g Database Concepts, Chap. 5.4]

Row Format: Oracle 10g

- "Firstly, Oracle rows are stored as a row header followed by column data. The row header contains a flag byte, lock byte and column count, then for each column there is a column length followed by the column data; column length is 1 or 3 bytes: 0-254 length: 1 bytes, else 3 bytes. To access the value of any column in a row, Oracle has to first examine the length bytes of all the preceding columns. This is a very quick and efficient operation, but it is done with such frequency that it nevertheless does have an impact on performance.
  ...
  Oracle normally requires one byte to represent each NULL, except that it does not store trailing NULLs in a data row."

- Source: (http://www.ixora.com.au/tips/table_column_order.htm)

Discussion

- Why a NULL Bitmap?
  - Allows to clearly distinguish between NULLs etc
  - Compact representation
  - Cf http://blogs.msdn.com/sqlserverstorageengine/archive/2006/06/29/650349.aspx
- Note that Oracle indicates NULLs with a field length of 0
  - Oracle cannot distinguish between NULL and an empty string!
  - Example:
    - CREATE TABLE test (id int, name varchar(10));
    - INSERT INTO test VALUES (1, 'string');
    - INSERT INTO test VALUES (2, '');
    - INSERT INTO test VALUES (3, NULL);
    - SELECT * FROM test WHERE name IS NOT NULL;
- Also technical note:
  - variable-length structures require ‘pointers’ and ‘length’ values to determine where next part starts;

References

- PostgreSQL 9
  - Row structure described in
    - http://www.postgresql.org/docs/9.0/static/storage-page-layout.html
    - http://www.postgresql.org/docs/9.0/static/catalog-pg-attribute.html
    - src/include/access/htup.h
    - src/backend/access/common/heaptuple.c
    - src/include/postgres.h
- SQL Server 2005
  - Row format of Express Version of SQL Server in online books
    - (SQL Server Database Engine: estimating table size)
  - Details of full version storage format in
    - http://blogs.msdn.com/sqlserverstorageengine/archive/2006/06/23/644607.aspx
- Oracle 10g
  - Row format described in *Oracle 10g Database Concepts*, Chapter 5.4