**week 4 CMU**

**Database Storage**

**DISK-ORIENTED ARCHITURE**

-The DBMS assumes that the primary storage location of the database is on non-volatile disk.The DBMS's components manage the movement of data between non-volatile and volatile storage.

SLOTTED PAGES

-The most common layout scheme is called slotted pages.The slot array maps "slots" to the tuples' starting position offsets.The header keeps track of:

→ The # of used slots

→ The offset of the starting location of the last slot used.

**LOG-STRUCTURED FILE ORGANIZATION**

-Instead of storing tuples in pages, the DBMS only stores log records.The system appends log records to the file of how the database was modified:

→ Inserts store the entire tuple.

→ Deletes mark the tuple as deleted.

→ Updates contain the delta of just the attributes that were modified

To read a record, the DBMS scans the log backwards and "recreates" the tuple to find what it needs.

Build indexes to allow it to jump to locations in the log.

Compaction coalesces larger log files into smaller files by removing unnecessary records.

------------------------------------------------------------------------------------------------------------------------

**TUPLE STORAGE**

A tuple is essentially a sequence of bytes.It's the job of the DBMS to interpret those bytes into attribute types and values.The DBMS's catalogs contain the schema information about tables that the system uses to figure out the tuple's layout.

**Data Representation**

**INTEGER/BIGINT/SMALLINT/TINYINT**

**→ C/C++ Representation**

**FLOAT/REAL vs. NUMERIC/DECIMAL**

**→ IEEE-754 Standard / Fixed-point Decimals**

**VARCHAR/VARBINARY/TEXT/BLOB**

**→ Header with length, followed by data bytes.**

**TIME/DATE/TIMESTAMP**

**→ 32/64-bit integer of (micro)seconds since Unix epoch**

**VARIABLE PRECISION NUMBERS**

Inexact, variable-precision numeric type that uses the "native" C/C++ types.

→ Examples: FLOAT, REAL/DOUBLE

Store directly as specified by IEEE-754.

Typically faster than arbitrary precision numbers but can have rounding errors…

ROUNDING EXAMPLE

#include <stdio.h>

int main(int argc, char\* argv[]) {

float x = 0.1;

float y = 0.2;

printf("x+y = %f\n", x+y);

printf("0.3 = %f\n", 0.3);

}

Output

x+y = 0.300000

0.3 = 0.300000

**FIXED PRECISION NUMBERS**

Numeric data types with (potentially) arbitrary precision and scale. Used when rounding errors are unacceptable.

→ Example: NUMERIC, DECIMAL

Many different implementations.

→ Example: Store in an exact, variable-length binary representation with additional meta-data.

→ Can be less expensive if you give up arbitrary precision.

POSTGRES:NUMERIC

typedef unsigned char NumericDigit;

typedef struct {

int ndigits;

int weight;

int scale;

int sign;

NumericDigit \*digits;

} numeric;

**MYSQL:NUMERIC**

typedef int32 decimal\_digit\_t;

struct decimal\_t {

int intg, frac, len;

bool sign;

decimal\_digit\_t \*buf;

};

**LARGE VALUES**

Most DBMSs don't allow a tuple to exceed the size of a single page.To store values that are larger than a page, the DBMS uses separate overflow storage pages.

→ Postgres: TOAST (>2KB)

→ MySQL: Overflow (>½ size of page)

→ SQL Server: Overflow (>size of page)

**EXTERNAL VALUE STORAGE**

Some systems allow you to store a really large value in an external file.Treated as a BLOB type.

→ Oracle: BFILE data type

→ Microsoft: FILESTREAM data type

The DBMS cannot manipulate the contents of an external file.

→ No durability protections.

→ No transaction protections

**System Catalogs**

A DBMS stores meta-data about databases in its internal catalogs.

→ Tables, columns, indexes, views

→ Users, permissions

→ Internal statistics

Almost every DBMS stores the database's catalog inside itself (i.e., as tables).

→ Wrap object abstraction around tuples.

→ specialized code for "bootstrapping" catalog tables.

You can query the DBMS’s internal INFORMATION\_SCHEMA catalog to get info about

the database.

→ ANSI standard set of read-only views that provide info about all the tables, views, columns, and procedures in a database

DBMSs also have non-standard shortcuts to retrieve this information.

**ACCESSING TABLE SCHEMA**

List all the tables in the current database:/\*SHOW TABLES;mysql\*/

SELECT \*

FROM INFORMATION\_SCHEMA.TABLES

WHERE table\_catalog = '<db name>';

**DATABASE WORKLOADS**

**On-Line Transaction Processing (OLTP)**

→ Fast operations that only read/update a small amount of data each time.

**On-Line Analytical Processing (OLAP)**

→ Complex queries that read a lot of data to compute aggregates.

**Hybrid Transaction + Analytical Processing**

→ OLTP + OLAP together on the same database instance

**OBSERVATION**

The relational model does not specify that we have to store all of a tuple's attributes together in a single page.

This may not actually be the best layout for some workloads…

**OLTP**

On-line Transaction Processing:

→ Simple queries that read/update a small amount of data that is related to a single entity in the database.This is usually the kind of application that people build first.

**OLAP**

On-line Analytical Processing:

→ Complex queries that read large portions of the database spanning multiple entities.

You execute these workloads on the data you have collected from your OLTP application(s).

**DATA Storage Models**

The DBMS can store tuples in different ways that are better for either OLTP or OLAP workloads.

We have been assuming the n-ary storage model(aka "row storage") so far this semester.

**N-ARY STORAGE MODEL(NSM)**

The DBMS stores all attributes for a single tuple contiguously in a page.

Ideal for OLTP workloads where queries tend to operate only on an individual entity and insert heavy workloads.

**SELECT** \* **FROM** useracct

**WHERE** userName = ?

**AND** userPass = ?

------------------------------------------------

**SELECT COUNT**(U.lastLogin),

**EXTRACT**(month **FROM** U.lastLogin) **AS** month

**FROM** useracct **AS** U

**WHERE** U.hostname **LIKE** '%.gov'

**GROUP BY EXTRACT**(month **FROM** U.lastLogin)

----------------------------------------------

**Advantages**

→ Fast inserts, updates, and deletes.

→ Good for queries that need the entire tuple.

**Disadvantages**

→ Not good for scanning large portions of the table and/or a subset of the attributes

---------------------------------------------------------------------

**DECOMPOSITION STORAGE MODEL(DSM)**

The DBMS stores the values of a single attribute for all tuples contiguously in a page.

→ Also known as a "column store"

Ideal for OLAP workloads where read-only queries perform large scans over a subset of the table’s attributes.

-------------------------------------------------------------------------

**TUPLE IDENTIFICATION**

**Choice #1: Fixed-length Offsets**

→ Each value is the same length for an attribute.

**Choice #2: Embedded Tuple Ids**

→ Each value is stored with its tuple id in a column.

**Advantages**

→ Reduces the amount wasted I/O because the DBMS only reads the data that it needs.

→ Better query processing and data compression (more on this later).

**Disadvantages**

→ Slow for point queries, inserts, updates, and deletes because of tuple splitting/stitching.

-----------------------------------------------------------------------------------------

**DSM SYSTEM HISTORY**

1970s: Cantor DBMS

1980s: DSM Proposal

1990s: SybaseIQ (in-memory only)

2000s: Vertica, VectorWise, MonetDB

2010s: Everyone

-------------------------------------------------------------------------------------------------

**CONCULSION**

The storage manager is not entirely independent from the rest of the DBMS.

It is important to choose the right storage model for the target workload:

→ OLTP = Row Store

→ OLAP = Column Store