CS525: Advanced Database Organization

Notes 3: Database Storage
Part II

Gerald Balekaki

Department of Computer Science

Illinois Institute of Technology

gbalekaki@iit.edu

June 30^{st} 2023

Slides: adapted from courses taught by Andy Pavlo, Carnegie Mellon University, Hector Garcia-Molina, Stanford, & Shun Yan Cheung, Emory University

Reading

- `Database Systems: The Complete Book, 2nd Edition,`
  - *Chapter 2: Data Storage*
- `Database System Concepts (6th/7th Edition)`
  - *Chapter 10 (6th)/ Chapter 13 (7th)*

Database Storage

- `Problem#1`: How the DBMS represents the database in files on disk
  - i.e., how to lay out data on disk.
- `Problem#2`: How the DBMS manages its memory and move data back-and-forth from disk.

Today's Agenda

- Data Representation
  - This determines how a DBMS stores the actual bits for individual attributes.
- System Catalogs
- Storage Models
- Modification of Tuples

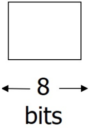
Tuple Storage

- A tuple is essentially a sequence of bytes (byte arrays).
- It is up to the DBMS to know how to interpret those bytes to derive the values for attributes.
- The DBMS's catalogs contain the schema information about tables that the system uses to figure out the tuple's layout.

What are the data items we want to store?

- a salary
- a name
- a date
- a picture

⇒ What we have available: Bytes



Data Representation

- How a DBMS stores the bytes for a value
- There are five high level data types that can be stored in tuples:
  - integers,
  - variable precision numbers,
  - fixed point precision numbers,
  - variable length values, and
  - dates/times.

Data Representation

How a DBMS stores the bytes for a value

- `INTEGER/BIGINT/SMALLINT/TINYINT`
  - `C/C++` Representation
    - All integers are stored in their "native" `C/C++` types.
- `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

Data Representation: Integers

- `C/C++` Representation
  - Most DBMSs store integers using their "native" `C/C++` types as specified by the IEEE-754 standard.
- These values are fixed length.
- Examples: `INTEGER/BIGINT/SMALLINT/TINYINT`

Variable Precision Numbers

- These are inexact[1], variable-precision numeric types that uses the "native" `C/C++` types.
- Store directly as specified by IEEE-754 standard.
- These values are also fixed length.
- Typically faster than arbitrary precision numbers because the CPU can execute instructions on them directly.
  - Example: `FLOAT, REAL/DOUBLE`
- but can have rounding errors[2] when performing computations due to the fact that some numbers cannot be represented precisely
  - To avoid this issue, we use Fixed-Point Precision Numbers.

[1] Inexact means that some values cannot be converted exactly to the internal format and are stored as approximations, so that storing and retrieving a value might show slight discrepancies.

[2] rounding error, is the difference between the result produced by a given algorithm using exact arithmetic and the result produced by the same algorithm using finite-precision, rounded arithmetic.

Variable Precision Numbers

- Rounding Example

```c
#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
```

Variable Precision Numbers

- Rounding Example

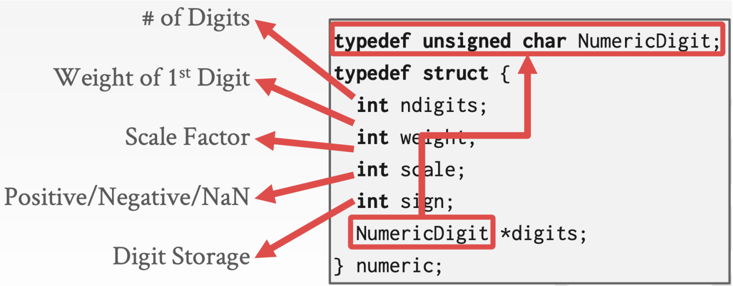
```c
#include <stdio.h>

int main(int argc, char* argv[]) {
   float x = 0.1;
   float y = 0.2;
   printf("x+y = %.20f\n", x+y);
   printf("0.3 = %.20f\n", 0.3);
}
```

```
Output
x+y = 0.30000001192092895508
0.3 = 0.29999999999999998890
```

Data Representation: Fixed Point Precision Numbers

- Numeric data types with arbitrary precision and scale.
- Used when round errors are unacceptable.
  - Example: `NUMERIC, DECIMAL`
- Typically stored in an exact, variable-length binary representation with additional meta-data that will tell the system things like the length of the data and where the decimal should be.
  - Like a `VARCHAR` but not stored as a string
- but the DBMS pays a performance penalty to get this accuracy.

PostgreSQL: NUMERIC



```c
/*
 * add_var() -
 *
 *  Full version of add functionality on variable level (handling signs).
 *  result might point to one of the operands too without danger.
 */
static void
add_var(const NumericVar *var1, const NumericVar *var2, NumericVar *result)
{
    /*
     * Decide on the signs of the two variables what to do
     */
    if (var1->sign == NUMERIC_POS)
    {
        if (var2->sign == NUMERIC_POS)
        {
            /*
             * Both are positive result = +(ABS(var1) + ABS(var2))
             */
            add_abs(var1, var2, result);
            result->sign = NUMERIC_POS;
        }
        else
        {
            /*
             * var1 is positive, var2 is negative Must compare absolute values
             */
            switch (cmp_abs(var1, var2))
            {
                case 0:
                    /* ----------
                     * ABS(var1) == ABS(var2)
                     * result = ZERO
                     * ----------
                     */
                    zero_var(result);
                    result->dscale = Max(var1->dscale, var2->dscale);
                    break;

                case 1:
                    /* ----------
                     * ABS(var1) > ABS(var2)
                     * result = +(ABS(var1) - ABS(var2))
                     * ----------
                     */
                    sub_abs(var1, var2, result);
                    result->sign = NUMERIC_POS;
                    break;

                case -1:
                    /* ----------
                     * ABS(var1) < ABS(var2)
                     * result = -(ABS(var2) - ABS(var1))
                     * ----------
                     */
                    sub_abs(var2, var1, result);
                    result->sign = NUMERIC_NEG;
                    break;
            }
        }
    }
    else
    {
        if (var2->sign == NUMERIC_POS)
        {
            /* ----------
             * var1 is negative, var2 is positive
             * Must compare absolute values
             * ----------
             */
            switch (cmp_abs(var1, var2))
            {
                case 0:
                    /* ----------
                     * ABS(var1) == ABS(var2)
                     * result = ZERO
                     * ----------
                     */
                    zero_var(result);
                    result->dscale = Max(var1->dscale, var2->dscale);
                    break;
```

PostgreSQL Source Code, numeric.c

Data Representation: Variable Length Data

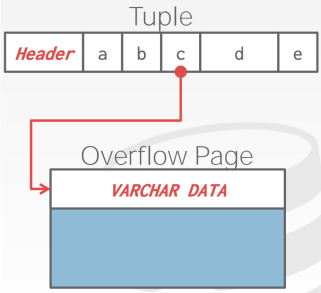
- These represent data types of arbitrary length.
  - An array of bytes of arbitrary length.
- Has a header that keeps track of the length of the string to make it easy to jump to the next value. It may also contain a checksum for the data.
- Example: `VARCHAR, VARBINARY, TEXT, BLOB`.

Large Values

- Most DBMSs don't allow a tuple to exceed the size of a single page.
- Two ways to handle that
  - overflow page
  - external storage

Large Values: Overflow Page

- To store values that are larger than a page, the DBMS uses separate overflow storage pages and have the tuple contain a reference to that page.
- Different DBMSs have different name/specification/requirements when they do that:
  - `Postgres: TOAST (The Oversized-Attribute Storage Technique) (>2KB)`
  - `MySQL: Overflow (>` $\frac{1}{2}$ `size of page)`
  - `SQL Server: Overflow (> size of page)`

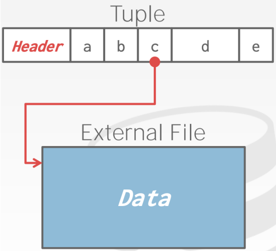


- These overflow pages can contain pointers to additional overflow pages until all the data can be stored.

External Value Storage

- Some systems allow to store a really large value in an external file and then the tuple will contain a pointer to that file.
- Example:
  - if the database is storing photo information, the DBMS can store the photos in the external files rather than having them take up large amounts of space in the DBMS.

- Treated as a `BLOB` type
  - `Oracle: BFILE data type`
    - Contains a locator to a large binary file stored outside the database.
  - `Microsoft: FILESTREAM data type`
- The DBMS **cannot** manipulate the contents of an external file.



- Reading: A paper explains the trade-offs between these two options:
  - To BLOB or Not To BLOB: Large Object Storage in a Database or a Filesystem

Data Representation: Dates and Times

- Varies widely across different database systems
- Usually, these are represented as the number of (micro/milli)seconds since the unix epoch.
- Example: `TIME, DATE, TIMESTAMP`.

System Catalog

- In order for the DBMS to be able to decipher the contents of tuples, it maintains an internal catalog to tell it meta-data about the databases
  - A DBMS stores meta-data about databases in its internal catalogs.
- The meta-data will contain what tables and columns the databases have along with their types and the orderings of the values.
  - Tables, columns, indexes, views
  - Users, permissions
  - Internal statistics
- Almost every DBMS stores their a database's catalog in itself in the format that they use for their tables
  - They use special code to "bootstrap" these catalog tables (wrap low-level access methods to access the catalog)

System Catalog

- 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 of the tables, views, columns, and procedures in a database.
- DBMSs also have non-standard shortcuts to retrieve this information.

Accessing Table Schema

- List all of the tables in the current database:

```
-- SQL-92
SELECT *
FROM INFORMATION_SCHEMA.TABLES
WHERE table_catalog = '<db name>';
```

```
\d;           -- Postgres
SHOW TABLES; -- MySQL
.tables;      -- SQLite
```

Accessing Table Schema

- List all of the columns in the *student* table:

```
-- SQL-92
SELECT *
FROM INFORMATION_SCHEMA.TABLES
WHERE table_name = 'student'
```

```
\d student;          -- Postgres
DESCRIBE student;   -- MySQL
.schema student;    -- SQLite
```

Today's Agenda

- Data Representation
- System Catalogs
- Storage Models
  - Ways to store tuples in pages

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
- There are many different workloads for database systems.
- By workload[1], we are referring to the general nature of requests a system will have to handle.

[1] *A database workload is a set of requests that have some common characteristics such as application, source of request, type of query, business priority, and/or performance objectives*

Wikipedia Example

```
CREATE TABLE useracct (
  userID INT PRIMARY KEY,
  userName VARCHAR UNIQUE,
  ⋮
);
```

```
CREATE TABLE pages (
  pageID INT PRIMARY KEY,
  title VARCHAR UNIQUE,
  latest INT
  ↳REFERENCES revisions (revID),
);
```

```
CREATE TABLE revisions (
  revID INT PRIMARY KEY,
  userID INT REFERENCES useracct (userID),
  pageID INT REFERENCES pages (pageID),
  content TEXT,
  updated DATETIME
);
```

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.

```sql
SELECT P.*, R.*
FROM pages AS P
   INNER JOIN revisions AS R
      ON P.latest = R.revID
WHERE P.pageID = ?
```

```sql
UPDATE useracct
SET lastLogin = NOW(),
   hostname = ?
WHERE userID = ?
```

```sql
INSERT INTO revisions
VALUES (?,?...,?)
```

OLTP: On-line Transaction Processing

- Fast, short running operations
- Simple queries that operate on single entity at a time
- Typically handle more writes than reads
- Repetitive operations
- Usually the kind of application that people build first
- Example
  - User invocations of Amazon (Amazon storefront).
    - Users can add things to their cart,
    - they can make purchases,
    - but the actions only affect their accounts.

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).

```
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)
```

OLAP: On-line Analytical Processing

- Long running, more complex queries
- Reads large portions of the database
- Analyzing and deriving new data from existing data collected on the OLTP side
- Example
  - Amazon computing the five most bought items over a one month period for these geographical locations.

HTAP: Hybrid Transaction + Analytical Processing

- A new type of workload which has become popular recently is HTAP, which is like a combination which tries to do `OLTP` and `OLAP` together on the same database.
- Watch HTAP Databases: What is New and What is Next - SIGMOD22-HTAP-Tutorial- June 2022

Data Storage Model

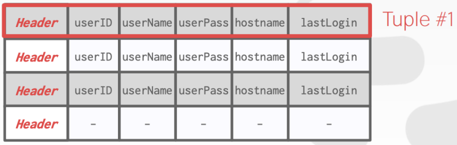
- There are different ways to store tuples in pages.
- The DBMS can store tuples in different ways that are better for either OLTP or OLAP workloads
- We have been assuming the n`-ary storage model` (aka "row storage") so far this semester.

N-ARY Storage Model (NSM)

- The DBMS stores all attributes for a single tuple contiguously in a single page.
- Ideal for OLTP workloads where requests are insert-heavy and transactions tend to operate only an individual entity
  - it takes only one fetch to be able to get all of the attributes for a single tuple.

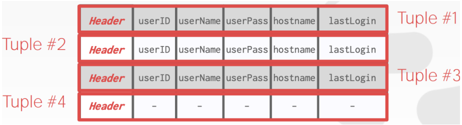
N-ARY Storage Model (NSM)

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



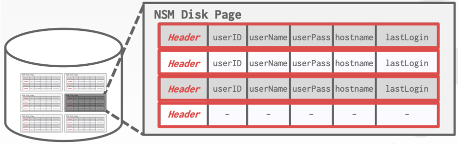
N-ARY Storage Model (NSM)

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



N-ARY Storage Model (NSM)

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

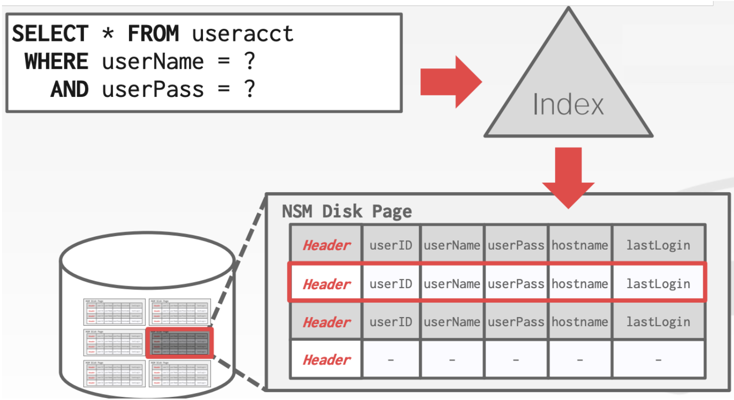


N-ARY Storage Model (NSM)

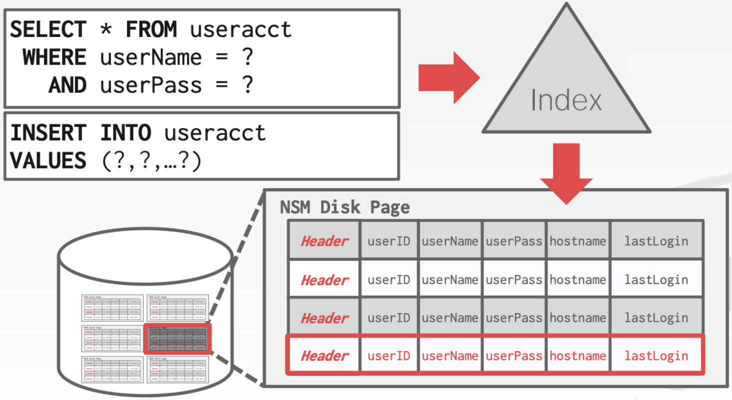
```sql
SELECT * FROM useracct
WHERE userName = ? AND userPass = ?
```

```sql
INSERT INTO useracct
VALUES (?,?,..,?)
```

N-ARY Storage Model (NSM)



N-ARY Storage Model (NSM)

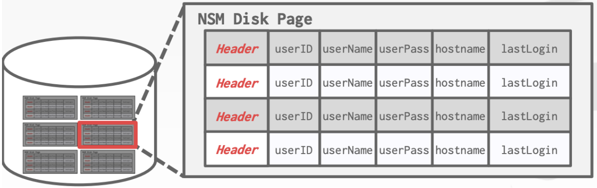


N-ARY Storage Model (NSM)

```sql
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)
```

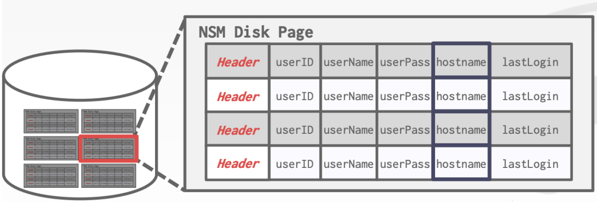
N-ARY Storage Model (NSM)

```sql
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)
```

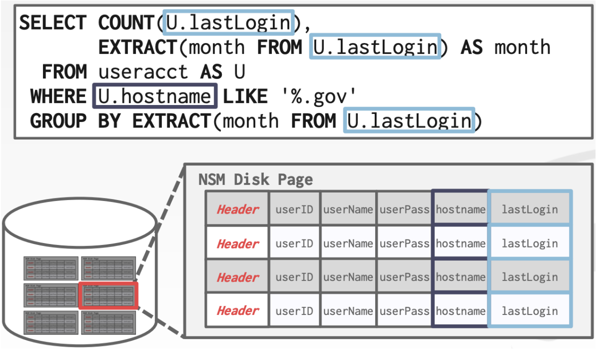


N-ARY Storage Model (NSM)

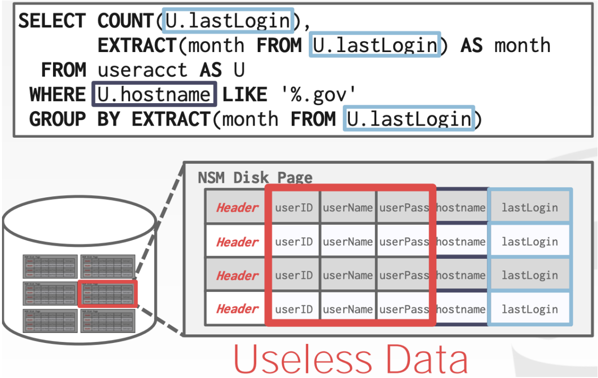
```sql
SELECT COUNT(U.lastLogin),
       EXTRACT(month FROM U.lastLogin) AS month
  FROM useracct AS U
 WHERE U.hostname LIKE '%.gov'
 GROUP BY EXTRACT(month FROM U.lastLogin)
```



N-ARY Storage Model (NSM)



N-ARY Storage Model (NSM)



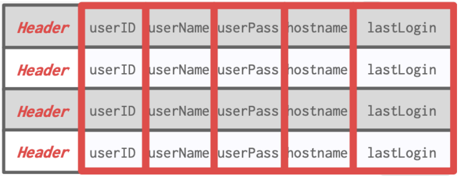
N-ARY Storage Model (NSM)

- 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.
  - This is because it pollutes the buffer pool by fetching data that is not needed for processing the query.

Decomposition Storage Model (DSM)

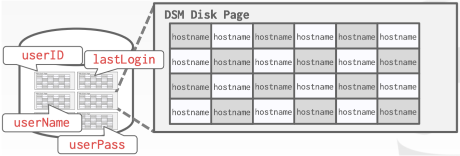
- The DBMS stores the values of a single attribute (column) for all tuples contiguously in a block of data.
  - *Vertically partition a database into a collection of individual columns that are stored separately*
  - Also known as a "column store".
- Ideal for OLAP workloads where read-only queries perform large scans over a subset of the table's attributes.

- The DBMS stores the values of a single attribute for all tuples contiguously in a page.
  - Also known as a “column store”.



Decomposition Storage Model (DSM)

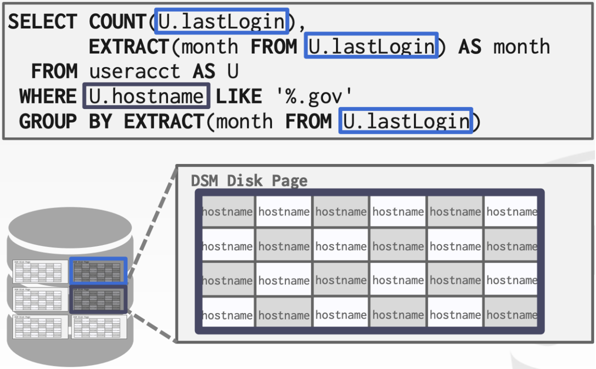
- The DBMS stores the values of a single attribute for all tuples contiguously in a page.
    - Also known as a “column store”.



Decomposition Storage Model (DSM)

```
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)
```

Decomposition Storage Model (DSM)



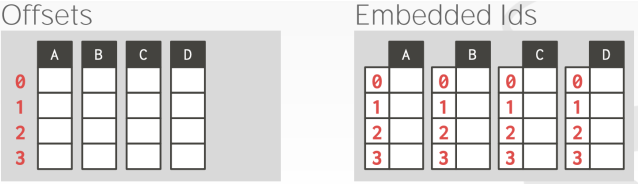
- *If there is a match on one page, how can we figure out a match on another page?*

Decomposition Storage Model: Tuple Identification

- To put the tuples back together when we are using a column store, we can use:
  - Choice #1: Fixed-length Offsets (most commonly used approach)
  - Choice #2: Embedded Tuple Ids (less common approach)

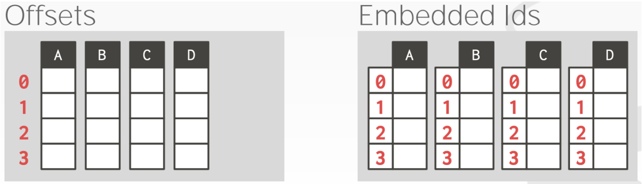
Choice #1: Fixed-length Offsets

- Assuming the attributes are all fixed-length, the DBMS can compute the offset of the attribute for each tuple.
- When the system wants the attribute for a specific tuple, it knows how to jump to that spot in the file from the offset.
- To accommodate the variable-length fields, the system can either pad fields so that they are all the same length or use a dictionary that takes a fixed-size integer and maps the integer to the value.



Choice #2: Embedded Tuple Ids

- A less common approach
- Each value is stored with its tuple id (ex: a primary key) in a column.
- The system also store a mapping to tell it how to jump to every attribute that has that id.
- Note that this method has a large storage overhead because it needs to store a tuple id for every attribute entry.



Decomposition Storage Model (DSM)

- Advantages
  - Reduces the amount wasted I/O because the DBMS only reads the data that it needs.
  - Enable better query processing and data compression.
    - because all of the values for the same attribute are stored contiguously
- Disadvantages
  - Slow for point queries, inserts, updates, and deletes because of tuple splitting/stitching.

Modification of Tuples

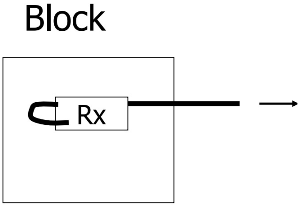
How to handle the following operations on the tuple level?

1. Insertion
2. Deletion
3. Update

1) Insertion

- **Easy case** Tuples fixed length/not in sequence(unordered)
  - Insert new tuple at end of file
  - or, in deleted slot
- **A little harder**
  - If records are variable size, not as easy
  - may not be able to reuse space - fragmentation
- **A Difficult case**: tuples in sequence (ordered)
  - Find position and slide following tuples
  - If tuples are sequenced by linking, insert overflow blocks

2) Deletion



Options

(a) Deleted and immediately reclaim space by shifting other tuples or removing overflows

(b) Mark deleted and list as free for re-use
  - May need chain of deleted tuples (for re-use)
  - Need a way to mark

Trade-offs

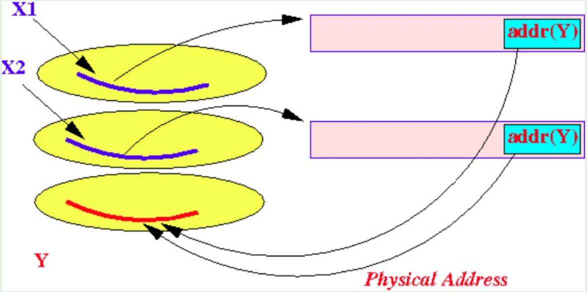
- How expensive is immediate reclaim?
  - How expensive is to move valid tuple to free space for immediate reclaim?
- How much space is wasted?
  - e.g., deleted tuples, delete fields, ...

Concern with deletions

A caveat when using physical addresses to reference a block/record

Example

- Record Y can be referenced by other tuples (e.g., tuples $X1$ & $X2$)

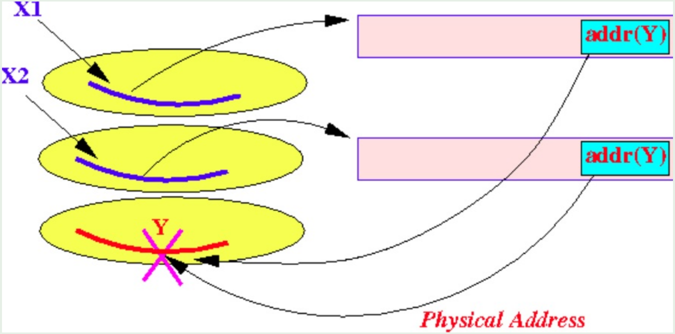


Concern with deletions

A caveat when using physical addresses to reference a block/record

Example

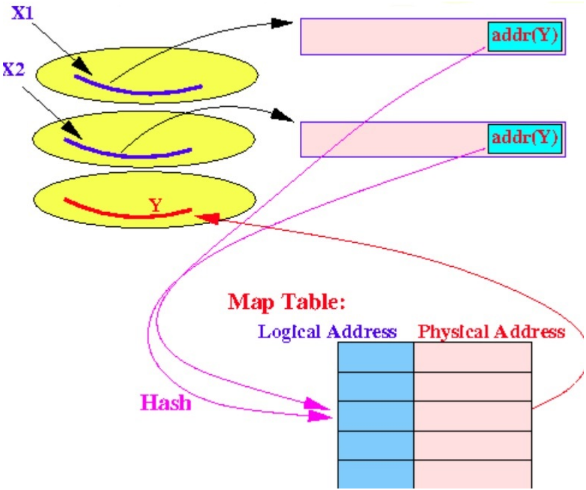
- When the tuple Y is deleted



- the physical addresses will reference an incorrect tuple

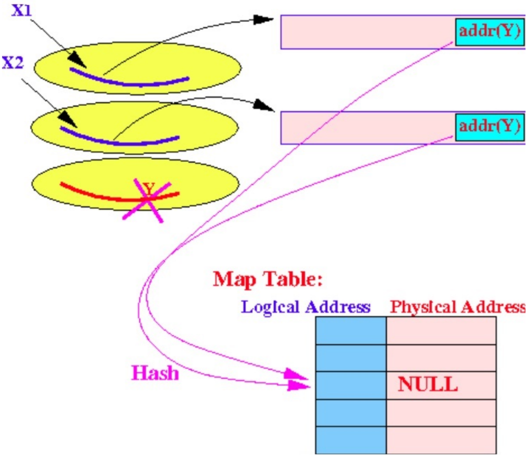
Techniques to handle tuple deletion

- Using logical addresses is easy
- Before deleting tuple Y that is referenced by tuples $X1$ and $X2$



Techniques to handle tuple deletion

- Using logical addresses is easy
- After deleting tuple Y



- Deleted tuple is identified by a NULL physical address in the Map table

Very important

- The logical address used by tuple Y must remain in the map table
- Furthermore:
  - The logical address used by tuple Y cannot be re-used

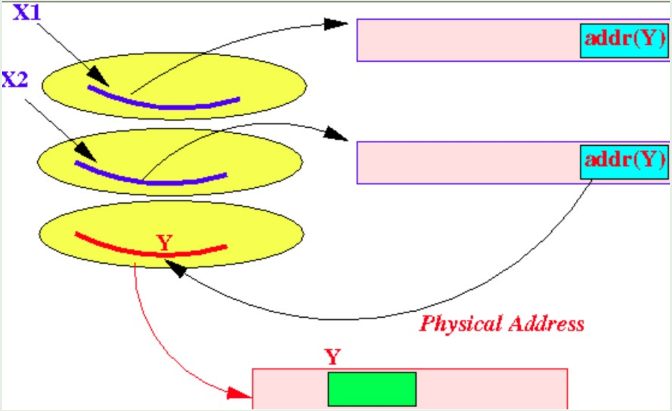
Techniques to handle tuple deletion

- Deleting a tuple using physical address: use a tombstone record
- Tombstone record: a (very small) special purpose tuple used to indicate a deleted tuple
- When a tuple is deleted, it is replaced by the tombstone record
- This tombstone is permanent, it must exist until the entire database is reconstructed
- Note: If we are using a map table, then the tombstone can be a null pointer in place of the physical address.

Tombstones

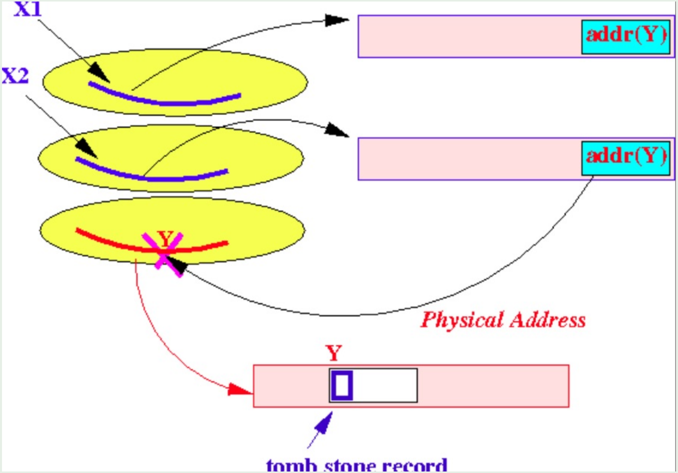
Example

- Before deleting tuple Y



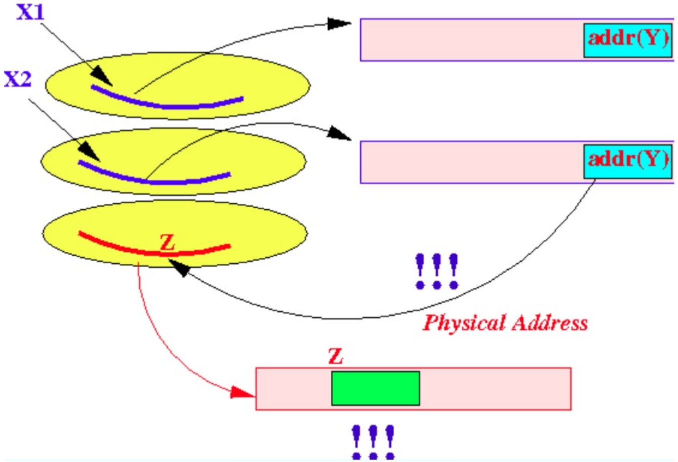
Example

- After deleting tuple Y



Tombstones

- When you insert a new tuple, you cannot use the space of a tombstone tuple (tombstone tuple must be preserved)
- Because: Existing tuple references to the deleted tuple will then references to the newly inserted tuple:



Update

- If new tuple is shorter than previous, easy
- If it is longer, need to shift tuples, create overflow blocks
- Note: We will never create a tombstone tuple in an update operation

Conclusion

- The storage manager is not entirely independent from the rest of the DBMS.
- A DBMS encodes and decodes the tuple's bytes into a set of attributes based on its schema.
- It is important to choose the right storage model for the target workload:
  - OLTP = Row Store
  - OLAP = Column Store

Database Storage: Next

- `Problem#1`: How the DBMS represents the database in files on disk
  - i.e., how to lay out data on disk.
- `Problem#2`: How the DBMS manages its memory and move data back-and-forth from disk.