>>

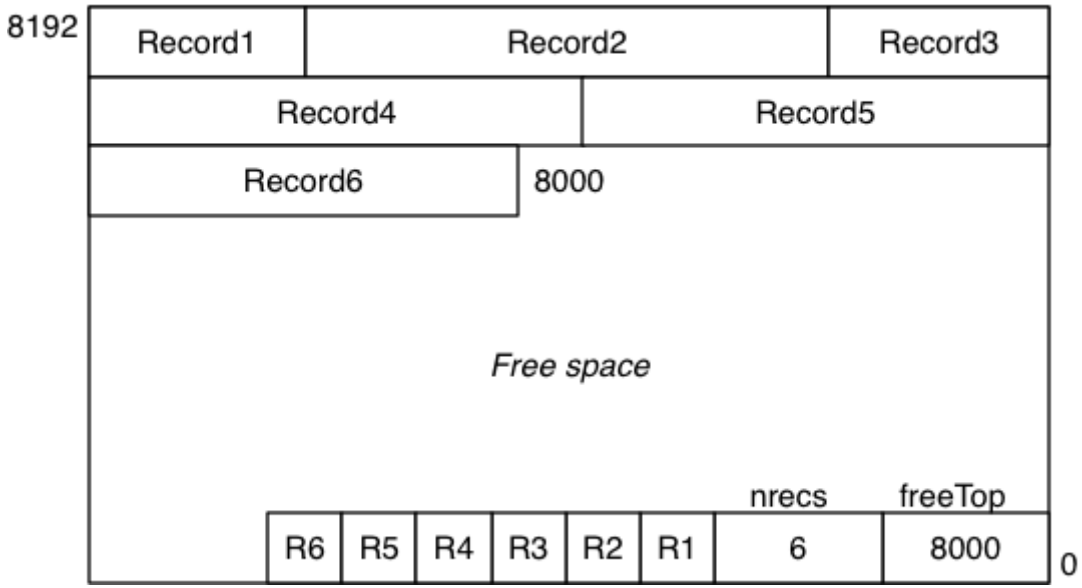
Tuple Representation

- Tuples
- Records vs Tuples
- Converting Records to Tuples
- Operations on Records
- Operations on Tuples
- Fixed-length Records
- Variable-length Records
- Data Types
- Field Descriptors

^ >>

❖ Tuples

Each page contains a collection of tuples



What do tuples contain? How are they structured internally?

<< ^ >>

❖ Records vs Tuples

A table is defined by a schema, e.g.

```
create table Employee (
    id   integer primary key,
    name varchar(20) not null,
    job  varchar(10),
    dept smallint references Dept(id)
);
```

where a schema is a collection of attributes (name,type,constraints)

Reminder: schema information (meta-data) is also stored, in the DB catalog

<< ^ >>

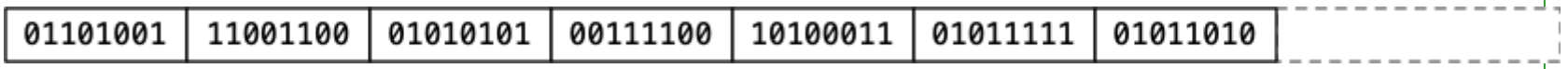
❖ Records vs Tuples (cont)

Tuple = collection of attribute values based on a schema, e.g.

```
(33357462,  'Neil Young',  'Musician',  277)
```

iid:integer | name:varchar(20) | job:varchar(10) | dept: smallint

Record = sequence of bytes, containing data for one tuple, e.g.



Bytes need to be interpreted relative to schema to get tuple

<< ^ >>

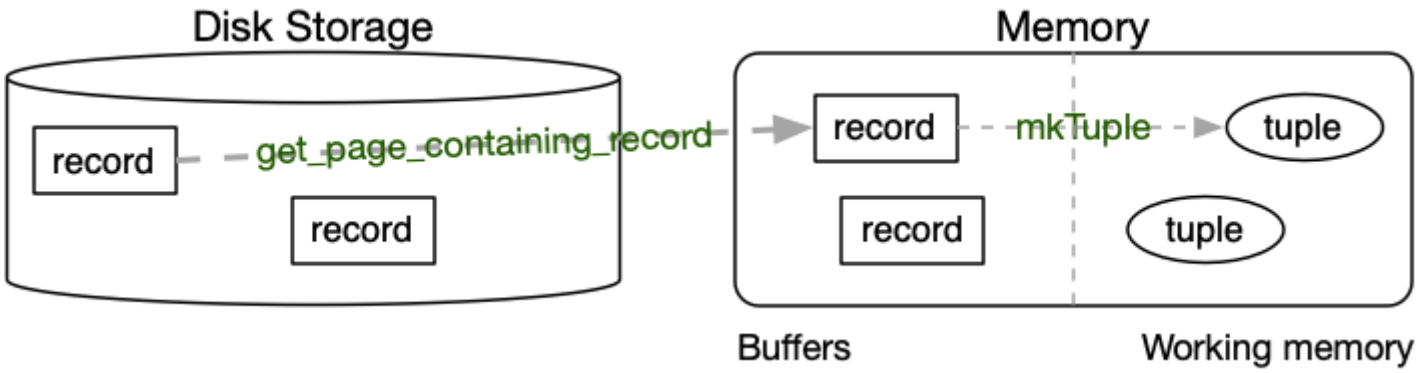
❖ Converting Records to Tuples

A **`Record`** is an array of bytes (**`byte[]`**)

- representing the data values from a typed **`Tuple`**
- stored on disk (persistent) or in a memory buffer

A **`Tuple`** is a collection of named,typed values (cf. C **`struct`**)

- to manipulate the values, need an "interpretable" structure
- stored in working memory, and temporary



<< ^ >>

❖ Converting Records to Tuples (cont)

Information on how to interpret bytes in a record ...

- may be contained in schema data in DBMS catalog
- may be stored in the page directory
- may be stored in the record (in a record header)
- may be stored partly in the record and partly in the schema

For variable-length records, some formatting info ...

- must be stored in the record or in the page directory
- at the least, need to know how many bytes in each varlen value

❖ Operations on Records

Common operation on records ... access record via **`RecordId`**:

```
Record get_record(Relation rel, RecordId rid) {
    (pid,tid) = rid;
    Page buf = get_page(rel, pid);
    return get_bytes(rel, buf, tid);
}
```

Cannot use a **`Record`** directly; need a **`Tuple`**:

```
Relation rel = ... // relation schema
Record rec = get_record(rel, rid)
Tuple t = mkTuple(rel, rec)
```

Once we have a **`Tuple`**, we can access individual attributes/fields

<< ^ >>

❖ Operations on Tuples

Once we have a record, we need to interpret it as a tuple ...

```
Tuple t = mkTuple(rel, rec)
```

- convert record to tuple data structure for relation **`rel`**

Once we have a tuple, we want to examines its contents ...

```
Typ   getTypField(Tuple t, int i)
```

- extract the **`i`**'th field from a **`Tuple`** as a value of type *Typ*

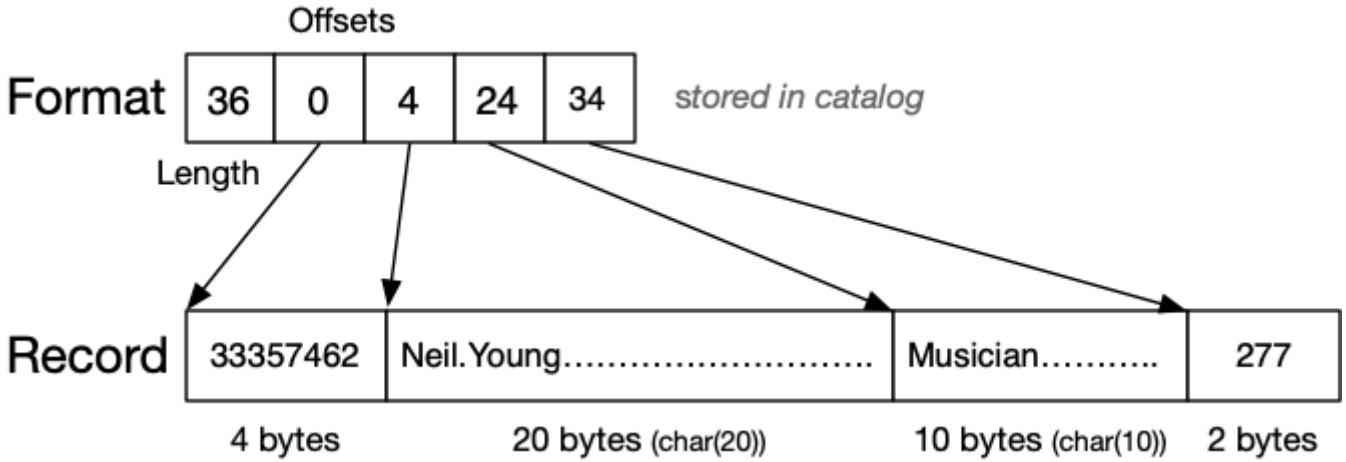
E.g. **`int x = getIntField(t,1)`**, **`char *s = getStrField(t,2)`**

<< ^ >>

❖ Fixed-length Records

A possible encoding scheme for fixed-length records:

- record format (length + offsets) stored in catalog
- data values stored in fixed-size slots in data pages



Since record format is frequently used at query time, cache in memory.

<< ^ >>

❖ Variable-length Records

Possible encoding schemes for variable-length records:

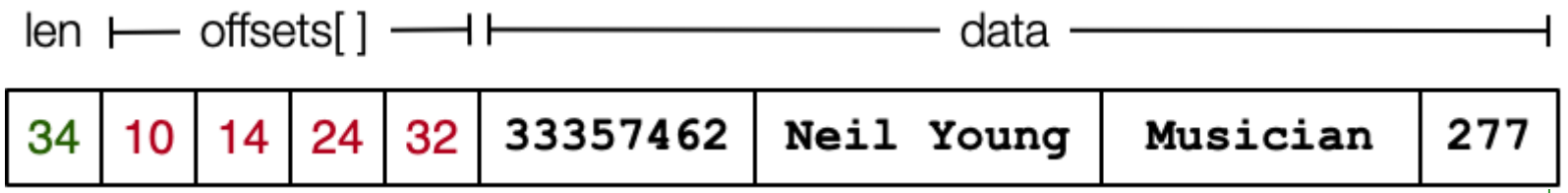
- Prefix each field by length

4	33357462	10	Neil Young	8	Musician	2	277

- Terminate fields by delimiter

33357462	X	Neil Young	X	Musician	X	277	X

- Array of offsets



<< ^ >>

❖ Data Types

DBMSs typically define a fixed set of base types, e.g.

DATE, **FLOAT**, **INTEGER**, **NUMBER(***n***)**, **VARCHAR(***n***)**, ...

This determines implementation-level data types for field values:

DATE	**time_t**
FLOAT	**float,double**
INTEGER	**int,long**
NUMBER(*n*)	**int[]** (?)
VARCHAR(*n*)	**char[]**

PostgreSQL allows new base types to be added

<< ^ >>

❖ Field Descriptors

A **`Tuple`** could be implemented as

- a list of field descriptors for a record instance
  (where a **`FieldDesc`** gives (offset,length,type) information)
- along with a reference to the **`Record`** data

```
typedef struct {
    ushort     nfields;    // number of fields/attrs
    ushort     data_off;   // offset in struct for data
    FieldDesc  fields[];   // field descriptions
    Record     data;       // pointer to record in buffer
} Tuple;
```

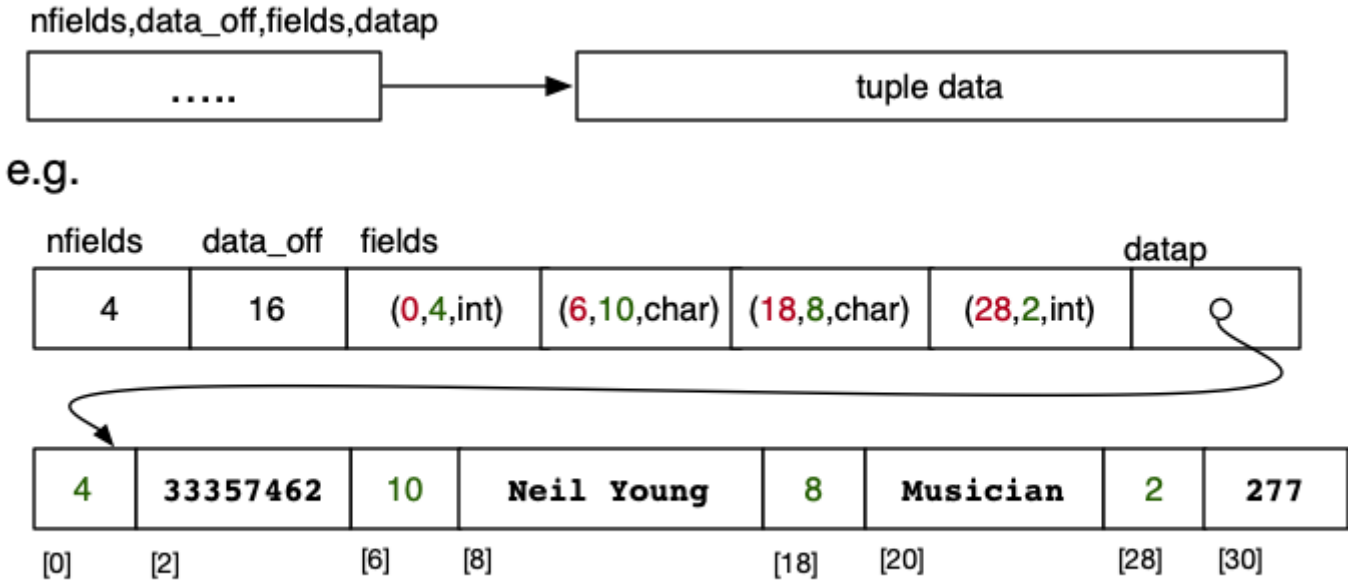
Fields are derived from relation descriptor + record instance data.

<< ^ >>

❖ Field Descriptors (cont)

Tuple **`data`** could be

- a pointer to bytes stored elsewhere in memory



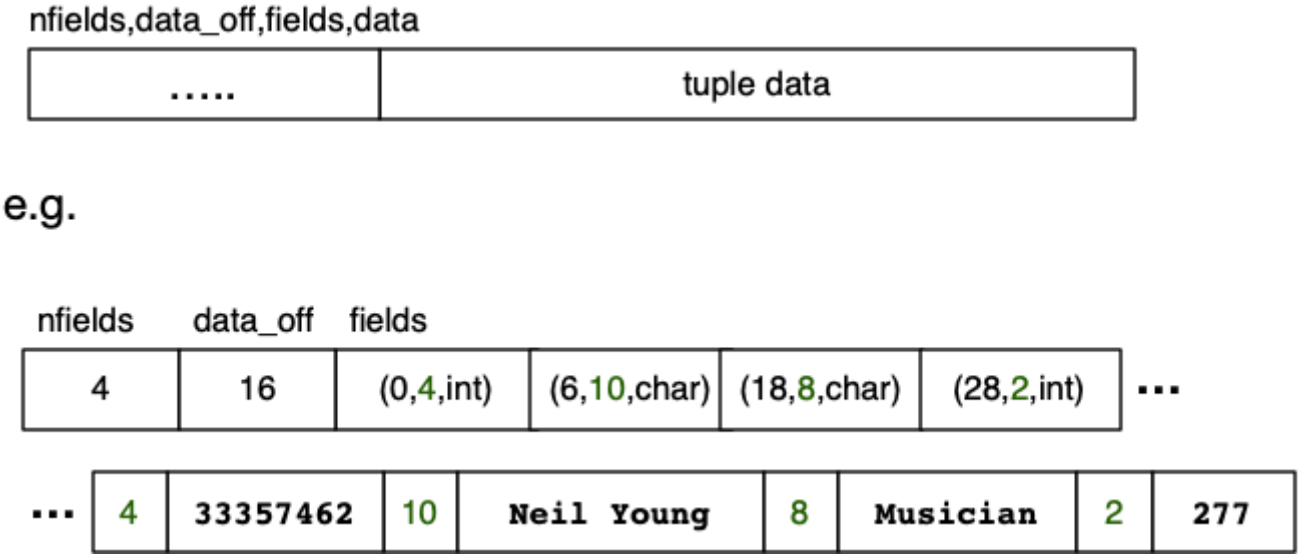
Note that the offset refers to the length field at the start of each attribute.

<< ^

❖ Field Descriptors (cont)

Or, tuple **`data`** could be ...

- appended to **`Tuple struct`** (used widely in PostgreSQL)



Produced: 27 Feb 2021