Revision
YOU choose the topics!

data modelling example

data types

capacity planning

joins

relational divide

server architecture

Data Modelling example

Extra data-modelling examples

- The Student Union wants a database to help in the management of student clubs. Each club has many members and any particular student can be a member of many clubs. Each club has a president and a treasurer. A student can be the president or treasurer in more than one club. Each club has a different membership fee.

- Each club has a clubroom where it can hold small meetings. Each clubroom is assigned to one club at most. The student union wants to keep track of which clubroom belongs to which club, but is not interested in keeping any information about the meetings held in the clubrooms.

- The clubrooms are too small for clubs to use for their annual general meetings (AGM), and so the Student Union also provides a small number of shared meeting rooms specifically for this purpose. Each club informs the Student Union of the date and time it wishes to hold its AGM, and the Union then chooses a meeting room that is free at that date and time that is large enough to hold that club's AGM.

- Initially, information about club's finances, assets, etc. will not be maintained in this database.

Data Types

Choosing data types

- Column: smallest unit of data in database
- Data types help DBMS to store and use information efficiently
- You should choose data types that:
  - enforce data integrity (quality)
  - can represent all possible values
  - support all required data manipulations
  - minimize storage space
  - maximize performance (e.g. fixed or variable length)
- The major data types are:
  - text
  - number
  - time

Character types (MySQL)

- **CHAR(M):** A fixed-length string that is always right-padded with spaces to the specified length when stored on disc. The range of M is 1 to 255.
- **CHAR:** Synonym for CHAR(1).
- **VARCHAR(M):** A variable-length string. Only the characters inserted are stored – no padding. The range of M is 1 to 65535 characters.
- **BLOB, TEXT:** A binary or text object with a maximum length of 65535 (2^16) bytes (blob) or characters (text). Not stored inline with row data.
- **LONGBLOB, LONGTEXT:** A BLOB or TEXT column with a maximum length of 4,294,967,295 (2^32 - 1) characters.
- **ENUM** (‘value1’,’value2’,…) up to 65,535 members.

Number types (MySQL)

- Integers
  - **TINYINT**: Signed (-128 to 127), Unsigned (0 to 255)
  - **BIT**, **BOOL:** synonyms for TINYINT
  - **SMALLINT**:
    Signed (-32,768 to 32,767), Unsigned (0 to 65,535 – 64k)
  - **MEDIUMINT**:
    Signed (-8388608 to 8388607), Unsigned (0 to 16777215 –16M)
  - **INT / INTEGER**:
    Signed (-2,147,483,648 to 2,147,483,647),
    Unsigned (0 to 4,294,967,295 – 4G or 2^32)
  - **BIGINT**:
    Signed (-9223372036854775808 to 9223372036854775807),
    Unsigned (0 to 18,446,744,073,709,551,615 - 2^64)
  - **Don't** use the "(M)" number for integers

Number types (MySQL)

- Real numbers (fractions)
  - **FLOAT**: single-precision floating point, allowable values: -3.402823466E+38 to -1.175494351E-38, 0, and 1.175494351E-38 to 3.402823466E+38.
  - **DOUBLE / REAL**: double-precision, allowable values: -1.7976931348623157E+308 to -2.2250738585072014E-308, 0, and 2.2250738585072014E-308 to 1.7976931348623157E+308.
  - optional M = number of digits stored, D = number of decimals.
  - Float and Double are often used for scientific data.
  - **DECIMAL**[(M[,D])]: fixed-point type. Good for money values.
  - M = precision (number of digits stored), D = number of decimals

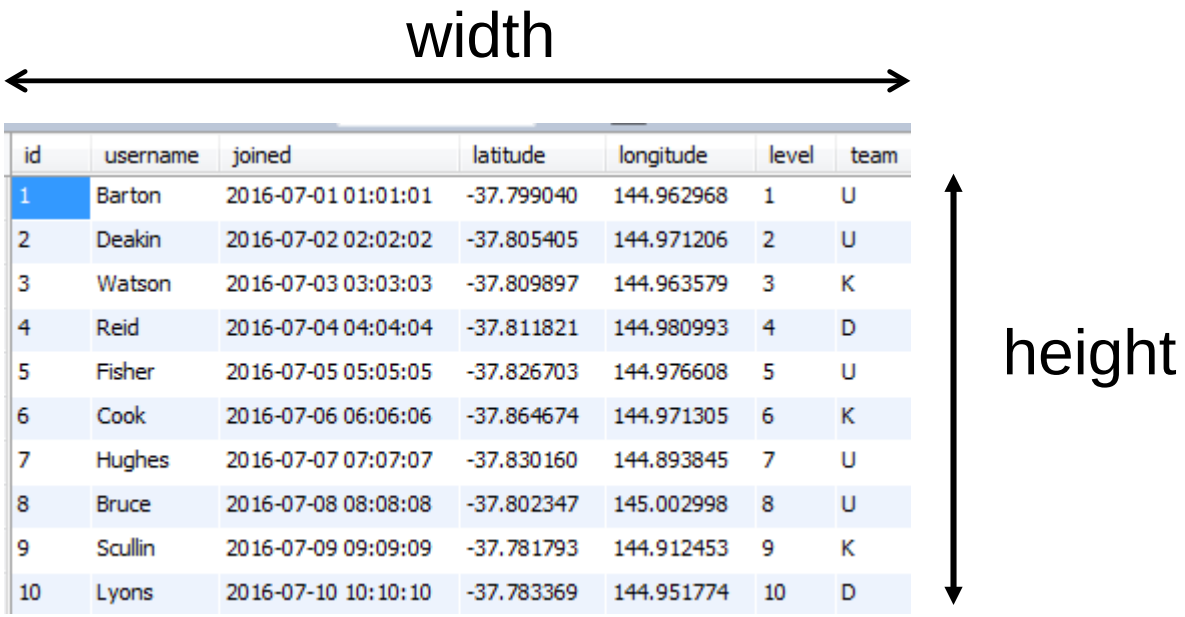
Time types

- **DATE** 1000-01-01 to 9999-12-31
- **TIME** -838:59:59 to 838:59:59
  (time of day or elapsed time)
- **DATETIME** 1000-01-01 00:00:00 to
  9999-12-31 23:59:59
- **TIMESTAMP** 1970-01-01 00:00:00 - ~ 2037
  Stored in UTC, converted to local
- **YEAR** 1901 to 2155

Capacity Planning

Estimating disk space requirements

- Which estimation methodology to use?
  - many vendors sell capacity planning solutions
  - most have the same ideas at their core
  - here we present the core concepts
- treat *Database size* as the sum of all *Table sizes*
  - where table size = number of rows * average row width



id	username	joined	latitude	longitude	level	team
1	Barton	2016-07-01 01:01:01	-37.799040	144.962968	1	U
2	Deakin	2016-07-02 02:02:02	-37.805405	144.971206	2	U
3	Watson	2016-07-03 03:03:03	-37.809897	144.963579	3	K
4	Reid	2016-07-04 04:04:04	-37.811821	144.980993	4	D
5	Fisher	2016-07-05 05:05:05	-37.826703	144.976608	5	U
6	Cook	2016-07-06 06:06:06	-37.864674	144.971305	6	K
7	Hughes	2016-07-07 07:07:07	-37.830160	144.893845	7	U
8	Bruce	2016-07-08 08:08:08	-37.802347	145.002998	8	U
9	Scullin	2016-07-09 09:09:09	-37.781793	144.912453	9	K
10	Lyons	2016-07-10 10:10:10	-37.783369	144.951774	10	D

Estimating row widths: numbers

- need to know storage size of different data types
- https://dev.mysql.com/doc/refman/8.0/en/storage-requirements.html

Numeric Type Storage Requirements

Data Type	Storage Required
TINYINT	1 byte
SMALLINT	2 bytes
MEDIUMINT	3 bytes
INT, INTEGER	4 bytes
BIGINT	8 bytes
FLOAT(*p*)	4 bytes if 0 <= *p* <= 24, 8 bytes if 25 <= *p* <= 53
FLOAT	4 bytes
DOUBLE [PRECISION], REAL	8 bytes
DECIMAL(*M*, *D*), NUMERIC(*M*, *D*)	Varies; see following discussion. Each multiple of nine digits requires four bytes,
BIT(*M*)	approximately (*M*+7)/8 bytes

Estimating row widths: dates and times

- (these sizes are for MySQL and are slightly different for other vendors)

Data Type	Storage Required Before MySQL 5.6.4	Storage Required as of MySQL 5.6.4
YEAR	1 byte	1 byte
DATE	3 bytes	3 bytes
TIME	3 bytes	3 bytes + fractional seconds storage
DATETIME	8 bytes	5 bytes + fractional seconds storage
TIMESTAMP	4 bytes	4 bytes + fractional seconds storage

Estimating row widths: text

String Type Storage Requirements

In the following table, *M* represents the declared column length in characters for nonbinary string types and bytes for binary string types. *L* represents the actual length in bytes of a given string value.

Data Type	Storage Required
CHAR(*M*)	The compact family of InnoDB row formats optimize storage for variable-length character sets. See COMPACT Row Format Storage Characteristics. Otherwise, *M* × *w* bytes, <= *M* <= 255, where *w* is the number of bytes required for the maximum-length character in the character set.
BINARY(*M*)	*M* bytes, 0 <= *M* <= 255
VARCHAR(*M*), VARBINARY(*M*)	*L* + 1 bytes if column values require 0 − 255 bytes, *L* + 2 bytes if values may require more than 255 bytes
TINYBLOB, TINYTEXT	*L* + 1 bytes, where *L* < 2^8
BLOB, TEXT	*L* + 2 bytes, where *L* < 2^{16}
MEDIUMBLOB, MEDIUMTEXT	*L* + 3 bytes, where *L* < 2^{24}
LONGBLOB, LONGTEXT	*L* + 4 bytes, where *L* < 2^{32}
ENUM('*value1*','*value2*',...)	1 or 2 bytes, depending on the number of enumeration values (65,535 values maximum)
SET('*value1*','*value2*',...)	1, 2, 3, 4, or 8 bytes, depending on the number of set members (64 members maximum)

Estimate growth of tables

For example: Using this simple database in which users post to forums, assume there are:

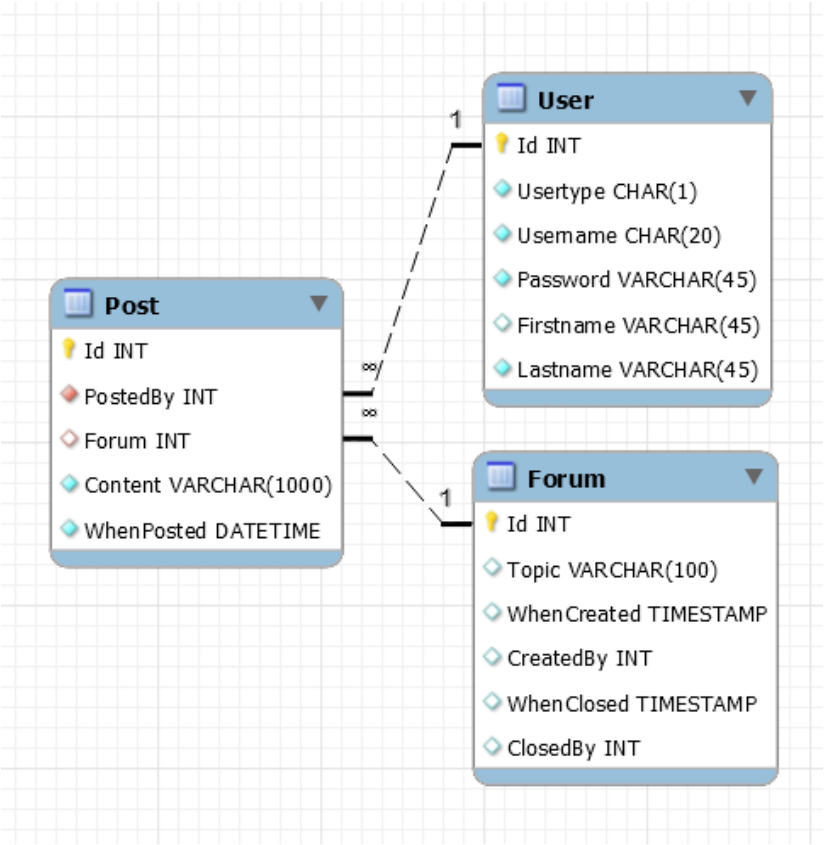
- 100 forums
- 1 million users

and assume that:

- users post on average 30 times per month

we calculate:

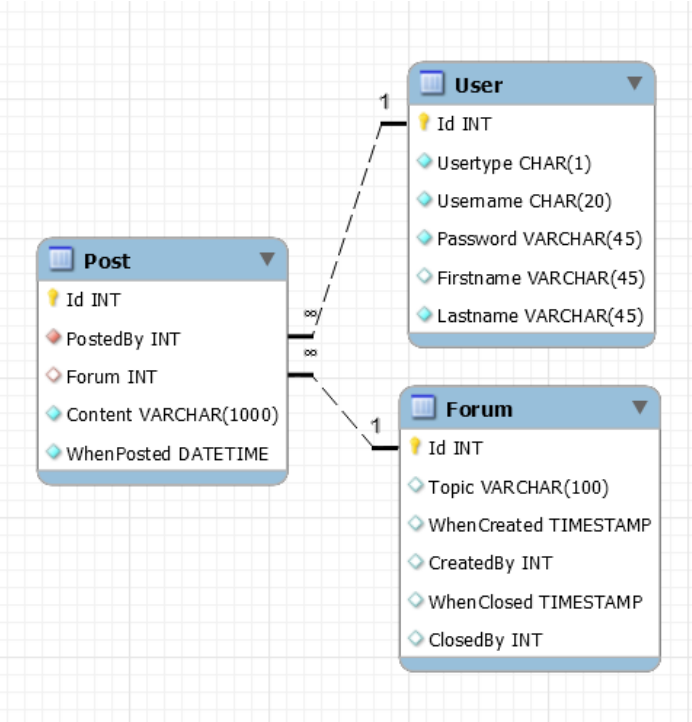
- *Post* table grows by 1M rows / day
- which means 12 Inserts per second



Calculate disk space per table

- use a spreadsheet to simplify calculations and enable what-ifs

column	type	width	rows	1 month	1 year
USER					
Id	int	4			
UserType	char(1)	1			
UserName	char(10)	10			
Password	char(10)	10			
FirstName	varchar(45)	12			
LastName	varchar(45)	15			
ROW WIDTH		**52**	1,000,000	1,100,000	2,000,000
DISK SPACE			52,000,000	57,200,000	**104,000,000**
FORUM					
Id	int	4			
Topic	varchar(100)	50		per month	
WhenCreated	timestamp	4		1	
CreatedBy	int	4			
ClosedBy	int	4			
ROW WIDTH		**66**	100	101	113
DISK SPACE			6,600	6,666	**7,458**
POST					
Id	bigint	8			
PostedBy	int	4		per user per month	months per year
Forum	int	4		30	12
Content	varchar(1000)	500			
WhenPosted	datetime	8			
ROW WIDTH		**524**	0	33,000,000	720,000,000
DISK SPACE			0	17,292,000,000	**377,280,000,000**



Joins

Inner join, Natural join

- Data about an entity is spread across 2 tables – so join them
- Inner/Equi join - Join rows where FK value = PK value

```
SELECT * FROM Customer INNER JOIN Account
    ON Customer.CustomerID = Account.CustomerID;
```

Export: Autosize:

CustomerID	CustFirstName	CustMiddleName	CustLastName	BusinessName	CustType	AccountID	AccountName	OutstandingBalance	CustomerID
1	Peter	NULL	Smith	NULL	Personal	1	Peter Smith	245.25	1
2	James	NULL	Jones	JJ Enterprises	Company	2	JJ ENt.	552.39	2
2	James	NULL	Jones	JJ Enterprises	Company	3	JJ ENt. Mgr	10.25	2

- Natural Join gives the same result as Inner Join
  - requires PK and FK columns to have the same name

```
SELECT * FROM Customer NATURAL JOIN Account;
```

Export: Autosize:

CustomerID	CustFirstName	CustMiddleName	CustLastName	BusinessName	CustType	AccountID	AccountName	OutstandingBalance
1	Peter	NULL	Smith	NULL	Personal	1	Peter Smith	245.25
2	James	NULL	Jones	JJ Enterprises	Company	2	JJ ENt.	552.39
2	James	NULL	Jones	JJ Enterprises	Company	3	JJ ENt. Mgr	10.25

Outer join

- Outer join
  - Can be left or right (see difference below)
  - Includes records from left/right table that don't have a matching row

```
SELECT * FROM Customer LEFT OUTER JOIN Account
    ON Customer.CustomerID = Account.CustomerID;
```

Export: Autosize:

CustomerID	CustFirstName	CustMiddleNa	CustLastName	BusinessName	CustType	AccountID	AccountName	OutstandingBalance	CustomerID
1	Peter	NULL	Smith	NULL	Personal	1	Peter Smith	245.25	1
2	James	NULL	Jones	JJ Enterprises	Company	2	JJ ENt.	552.39	2
2	James	NULL	Jones	JJ Enterprises	Company	3	JJ ENt. Mgr	10.25	2
3	Akin	NULL	Smithies	Bay Wart	Company	NULL	NULL	NULL	NULL

```
SELECT * FROM Customer RIGHT OUTER JOIN Account
    ON Customer.CustomerID = Account.CustomerID;
```

Export: Autosize:

CustomerID	CustFirstName	CustMiddleName	CustLastName	BusinessName	CustType	AccountID	AccountName	OutstandingBalance	CustomerID
1	Peter	NULL	Smith	NULL	Personal	1	Peter Smith	245.25	1
2	James	NULL	Jones	JJ Enterprises	Company	2	JJ ENt.	552.39	2
2	James	NULL	Jones	JJ Enterprises	Company	3	JJ ENt. Mgr	10.25	2

Cartesian join

- What if there is no join condition?

```
SELECT * FROM Customer, Account;
```

Export: Autosize:

CustomerID	CustFirstName	CustMiddleName	CustLastName	BusinessName	CustType	AccountID	AccountName	OutstandingBalance	CustomerID
1	Peter	NULL	Smith	NULL	Personal	1	Peter Smith	245.25	1
2	James	NULL	Jones	JJ Enterprises	Company	1	Peter Smith	245.25	1
3	Akin	NULL	Smithies	Bay Wart	Company	1	Peter Smith	245.25	1
1	Peter	NULL	Smith	NULL	Personal	2	JJ ENt.	552.39	2
2	James	NULL	Jones	JJ Enterprises	Company	2	JJ ENt.	552.39	2
3	Akin	NULL	Smithies	Bay Wart	Company	2	JJ ENt.	552.39	2
1	Peter	NULL	Smith	NULL	Personal	3	JJ ENt. Mgr	10.25	2
2	James	NULL	Jones	JJ Enterprises	Company	3	JJ ENt. Mgr	10.25	2
3	Akin	NULL	Smithies	Bay Wart	Company	3	JJ ENt. Mgr	10.25	2

NOT CORRECT: lack of join conditions -> Cartesian product
(every row in Customer combined with every record in Account)

Referential Integrity

Referential Integrity

```
CREATE TABLE Customer (
   CustomerID         smallint                        auto_increment,
   CustFirstName      varchar(100),
   CustMiddleName     varchar(100),
   CustLastName       varchar(100)                    NOT NULL,
   BusinessName       varchar(200),
   CustType           enum('Personal','Company')      NOT NULL,
   PRIMARY KEY  (CustomerID)
) ENGINE=InnoDB;
```

```
CREATE TABLE Account (
   AccountID             smallint        auto_increment,
   AccountName           varchar(100)    NOT NULL,
   OutstandingBalance    DECIMAL(10,2)   NOT NULL,
   CustomerID            smallint        NOT NULL,
   PRIMARY KEY  (AccountID),
   FOREIGN KEY (CustomerID) REFERENCES Customer(CustomerID)
         ON DELETE RESTRICT
         ON UPDATE CASCADE
) ENGINE=InnoDB;
```

Referential Actions how foreign keys guarantee referential integrity.

Flow control using CASE

Java Switch Statements

Use the `switch` statement to select one of many cod

Syntax

```
switch(expression) {
  case x:
    // code block
    break;
  case y:
    // code block
    break;
  default:
    // code block
}
```

VB

```
Dim number As Integer = 8
Select Case number
    Case 1 To 5
        Debug.WriteLine("Between 1 and 5, inclusive")
        ' The following is the only Case clause that evaluates to True.
    Case 6, 7, 8
        Debug.WriteLine("Between 6 and 8, inclusive")
    Case 9 To 10
        Debug.WriteLine("Equal to 9 or 10")
    Case Else
        Debug.WriteLine("Not between 1 and 10, inclusive")
End Select
```

Flow Control using CASE

- A better solution is to use the CASE expression

```
UPDATE Salaried
    SET AnnualSalary =
        CASE
            WHEN AnnualSalary <= 100000
            THEN AnnualSalary * 1.05
            ELSE AnnualSalary * 1.10
        END;
```

- now we process each row independently, one at a time

Flow Control using CASE

- CASE can also be used in SELECT statements
- e.g “Calculate our annual bonuses. Give each employee a 10% bonus, except those who work work in Clothes or Books, who get 20%.”

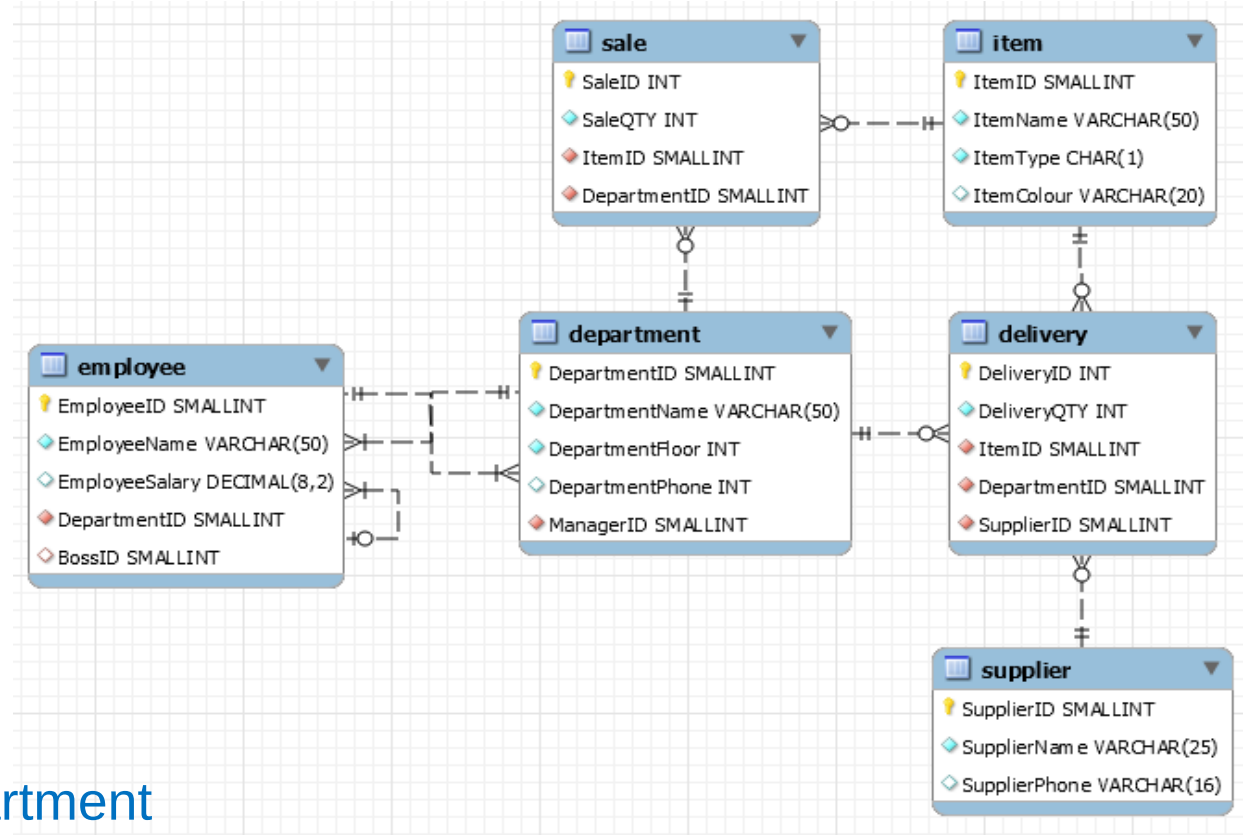
```
SELECT employeeId, lastName, departmentId, salary,
CASE
    WHEN departmentId in
        (SELECT departmentId FROM Department WHERE name in ('clothes', 'books'))
    THEN salary * 0.2
    ELSE salary * 0.1
END as bonus
FROM employee
ORDER BY departmentid;
```

Result Grid

employeeId	lastName	departmentId	salary	bonus
1	Munro	1	125000.00	12500.000
11	Skeeter	2	45000.00	9000.000
13	Smith	3	46000.00	9200.000
12	Montez	3	46000.00	9200.000
15	Mason	4	45000.00	4500.000
14	Innit	4	41000.00	4100.000

Relational Divide

SQL: relational divide




```
SELECT * FROM ITEM
WHERE NOT EXISTS
  (SELECT * FROM Department
   WHERE NOT EXISTS
     (SELECT * FROM Delivery
      WHERE Delivery.itemid = Item.itemid
      AND Delivery.departmentid = Department.departmentid ) );
```

Server Architecture

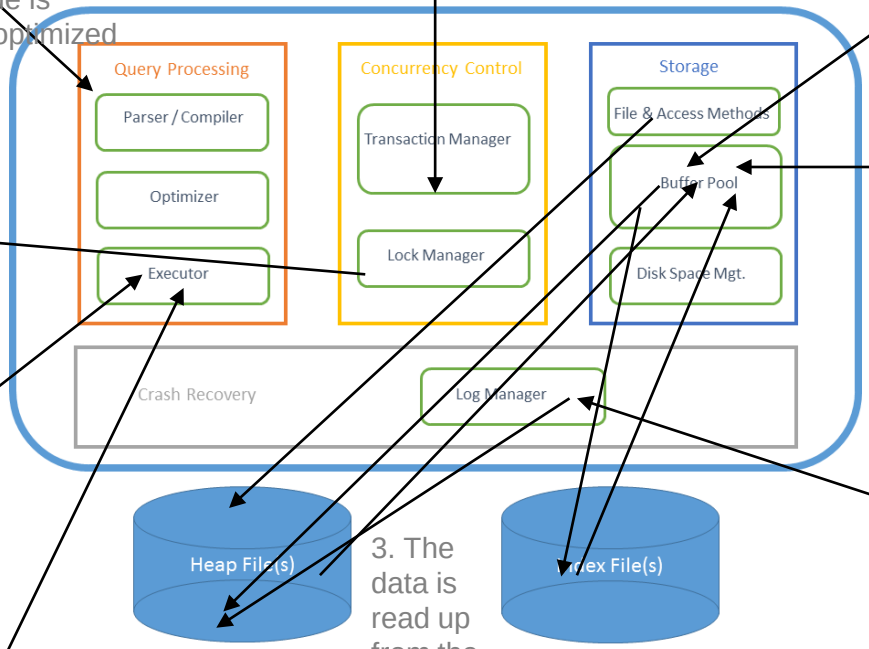
How the DBMS processes a query

```
UPDATE Employee
SET DepartmentID = 9
WHERE EmployeeName = 'Nancy'
AND DepartmentID = 8;
```

1. The code is parsed & optimized

4. The Concurrency manager ensures all the tables and index data is in the buffer pool and ready to be used in the execution of the SQL.

6a In the buffer pool we record the before value the after value and the change to the value



2. We check if the table & index data (rows) are in the buffer pool. If it is we skip to 4. If not we go to 3.

5a When the tables & indexes are locked we execute the query

5b. The SQL is executed

3. The data is read up from the table & index on disk

6b We record a timestamp and the before value, the after value and the change to the value for all transactions in the Log Manager

7. The user commits the change to make it permanent

8. The changed data and transaction log are written to disk

Graphic Courtesy of
Dr Renata Borovica-Gajic