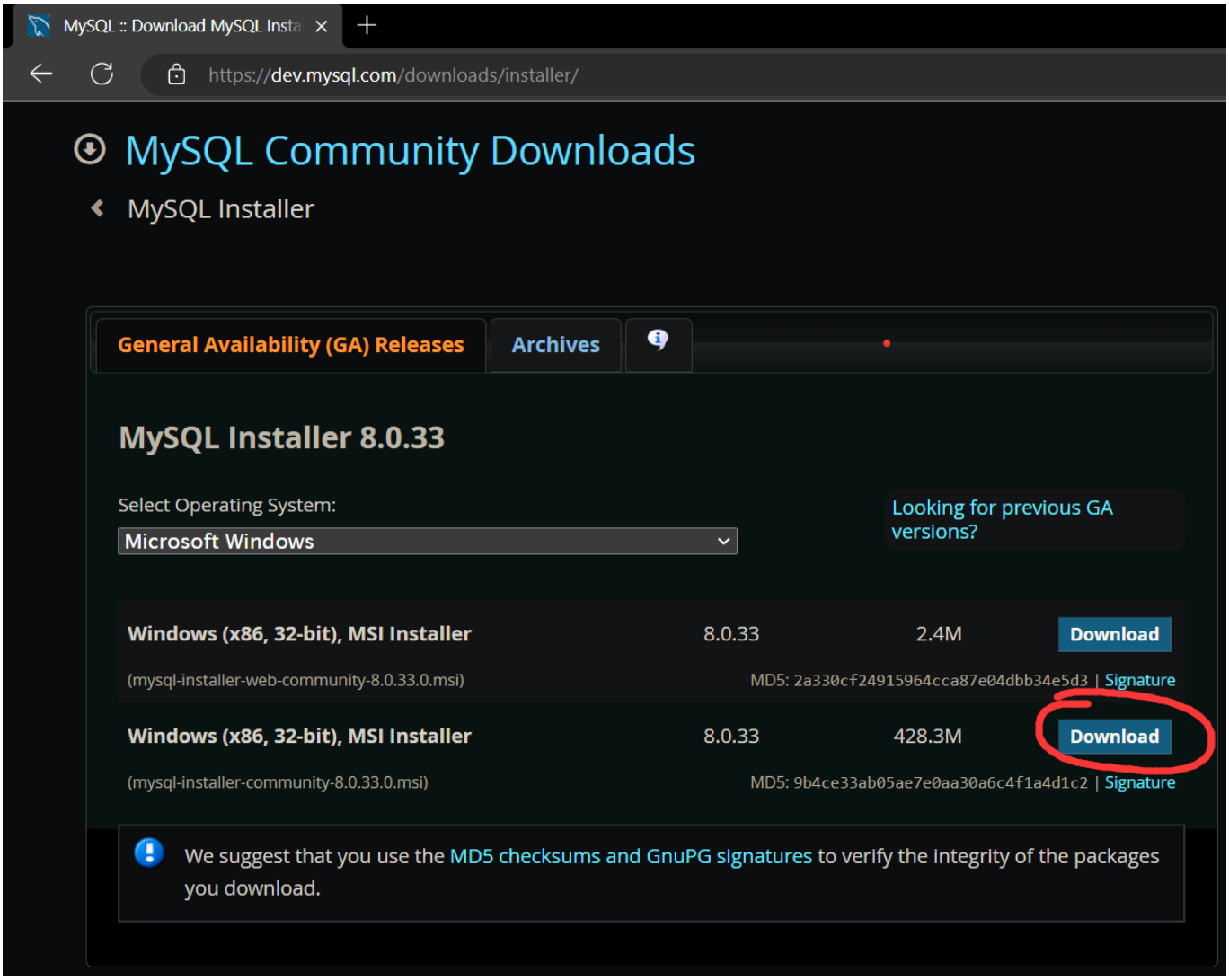
In this course we use data from dvdrental.sql

Content

Pre-requirements for Windows

Download and Install Mysql

Download Mysql from https://dev.mysql.com/downloads/installer/



do not select the web version

follow the instructions (default, default, default...)

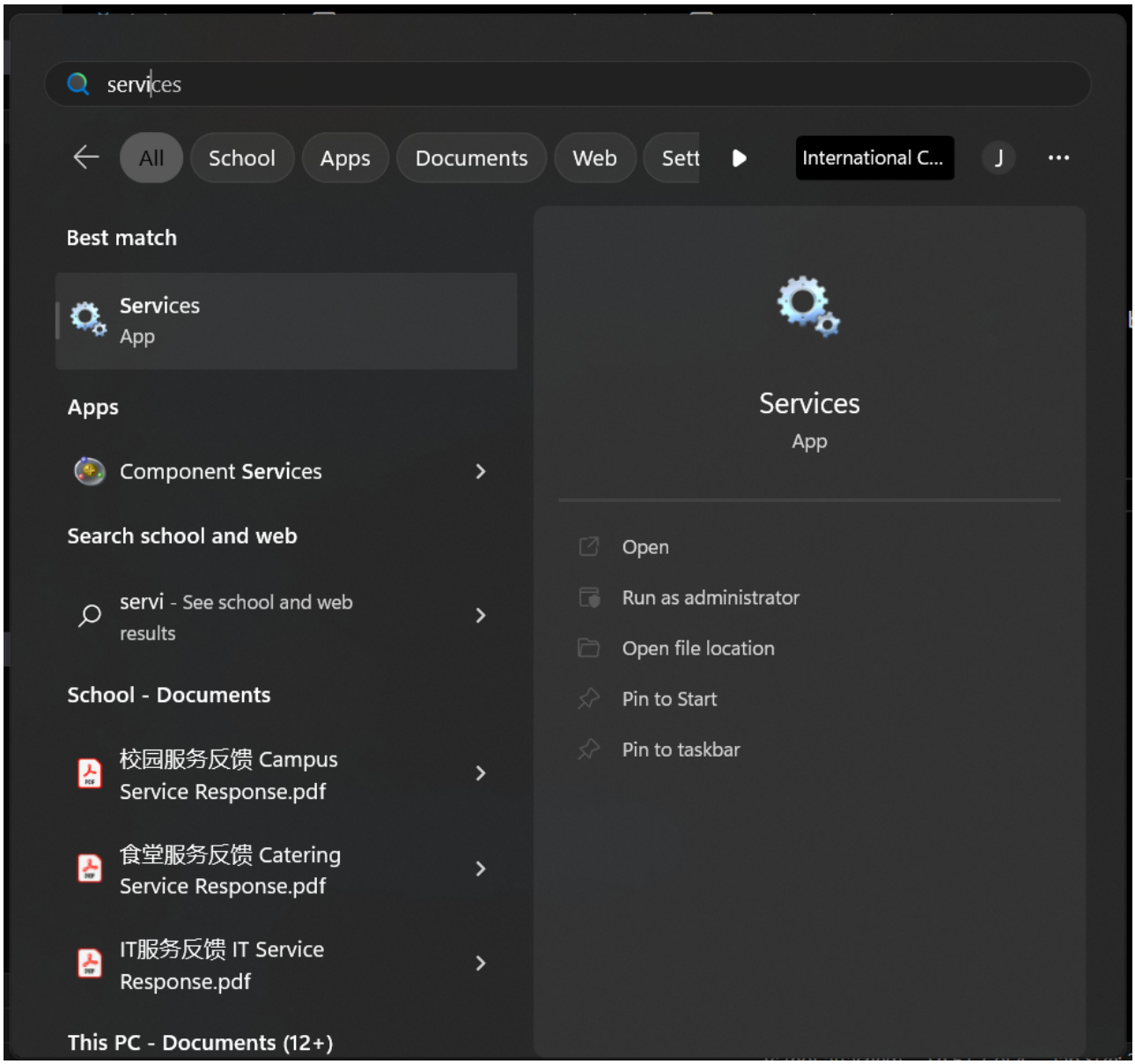
Optional* Download and Install DBeaver

Download DBeaver from https://dbeaver.io/. It's like an IDE for databases which greatly eaiser the editing of sql and database.

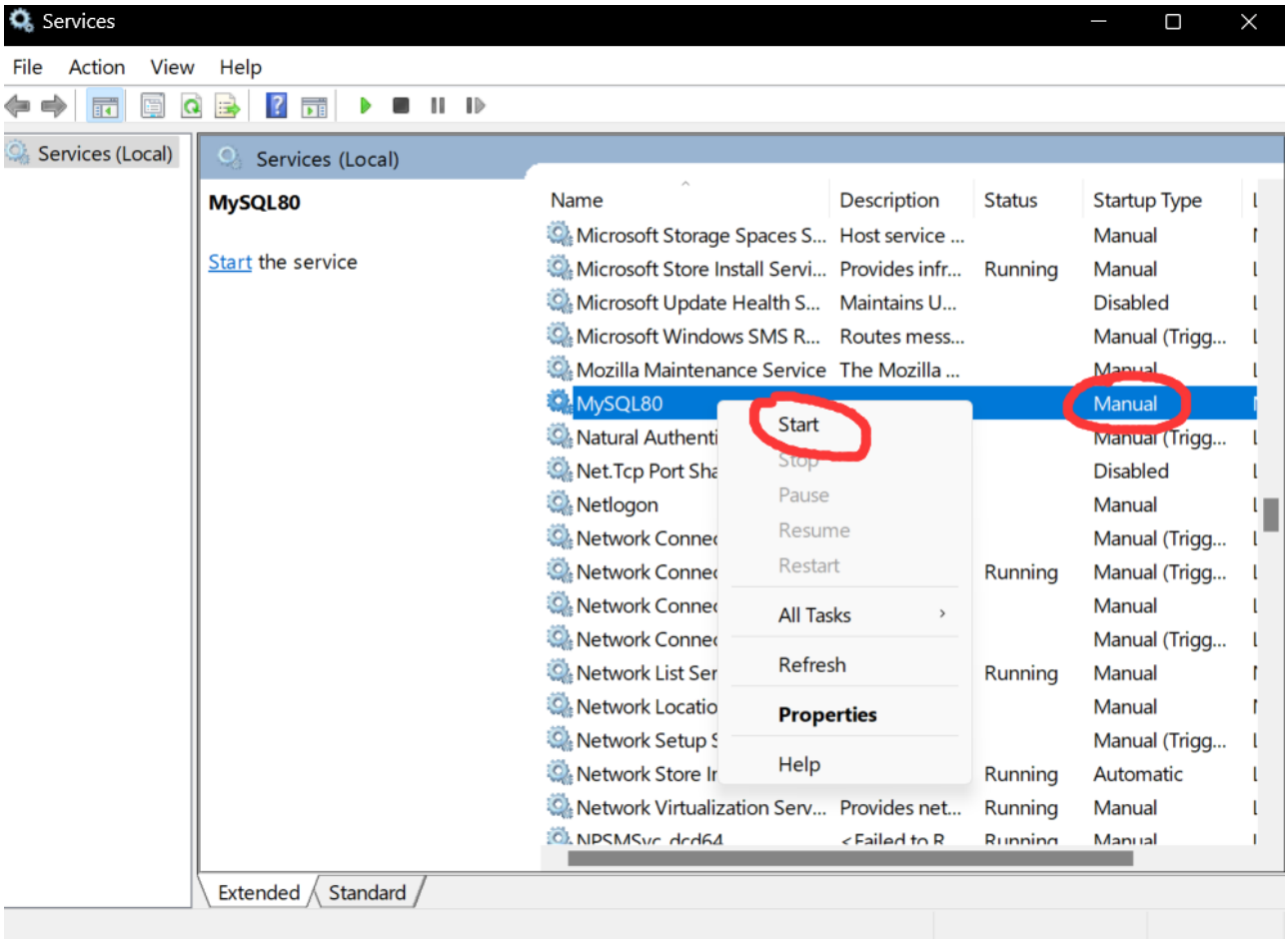
Open or stop Mysql service

The are two ways:

1. Through GUI (recommended for noobs)

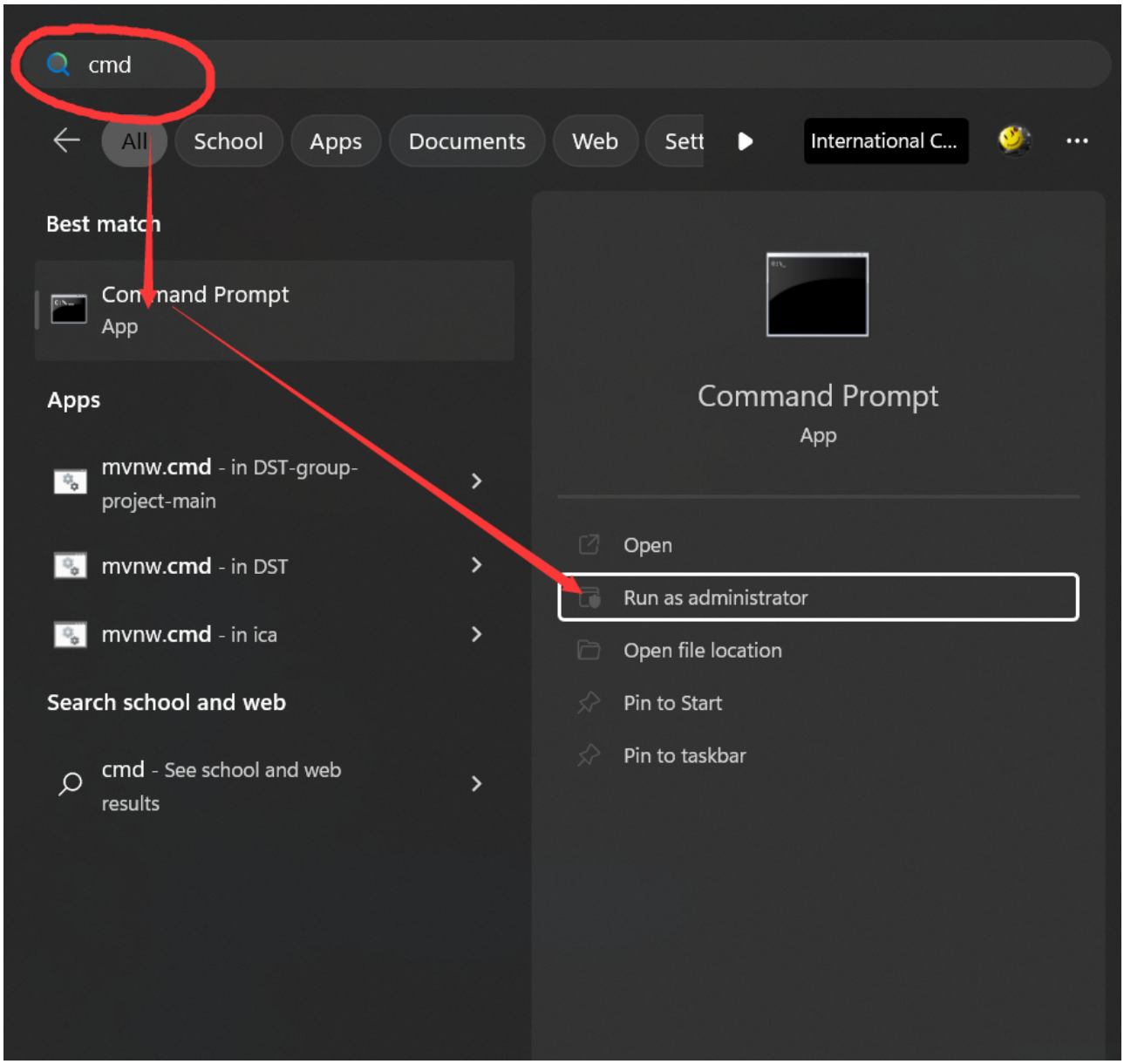


*find **Services***



set as Manual and start service (stop service to reduce performance cost)

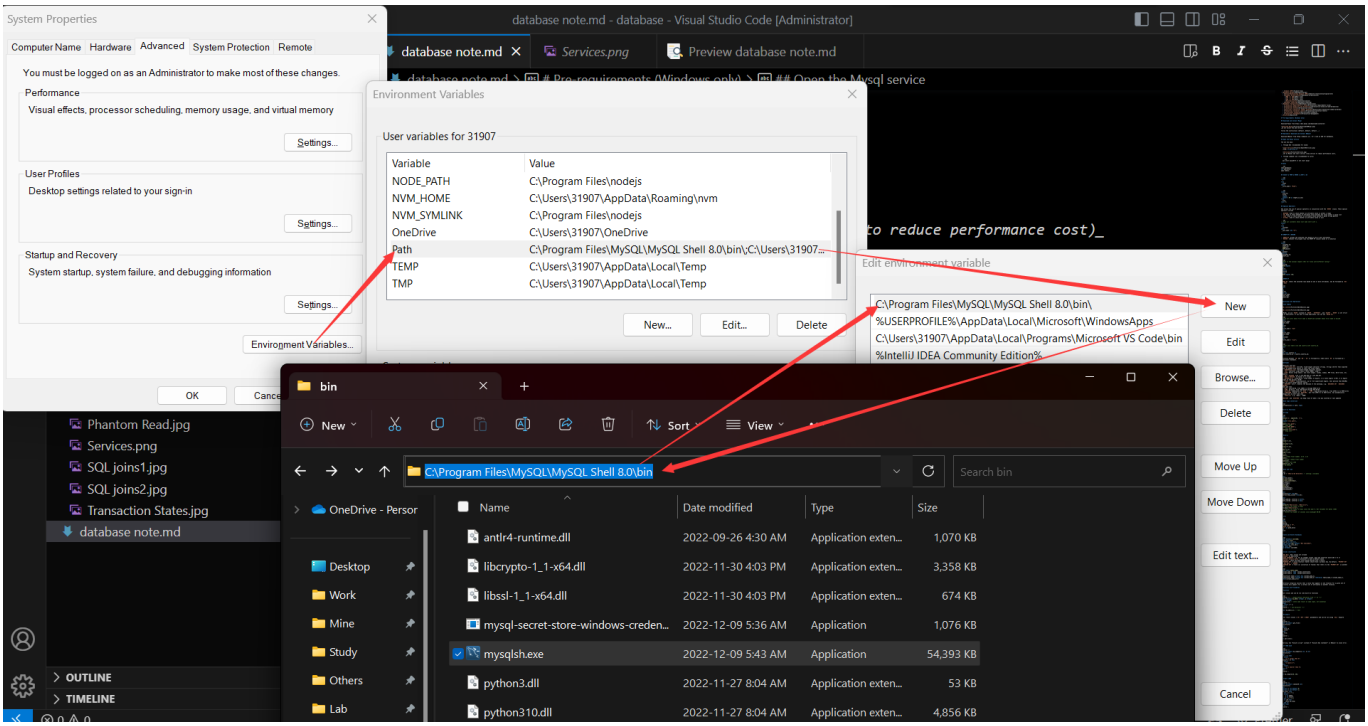
2. Through command line (recommended for pros)



Run cmd as administrator

```
# To enable mysql service, type
net start mysql80
# if it doesn't work, type
net start mysql
# To stop mysql service, type
net start mysql80
# if it doesn't work, type
net stop mysql
```

Try Mysql in command line



add Mysql to Path

```
# To enter mysql mode, type
mysql
# If it doesn't work, type
mysqlsh
# To exit mysql mode, type \quit
```

Query

```
show databases;
use dvdrental;
show tables;
```

SELECT & FROM & WHERE & LIMIT & AS

```
select
    *
from
    staff
where
    first_name = "Mike";
```

```
select
    film_id,
```

```
    title,
    length,
    length * 60 as length_in_secs
from
    film
limit 5;
```

Special Operators

SQL allows the use of special operators in conjunction with the `WHERE` clause. These special operators include

- `BETWEEN` Used to check whether an attribute value is within a range
- `IN` Used to check whether an attribute value matches any value within a value list
- `LIKE` Used to check whether an attribute value matches a given string pattern
- `IS NULL` Used to check whether an attribute value is null

```
-- Find out customers whose last name start with J.
select
    *
from
    customer
where
    last_name like "J%";
```

GROUP BY & HAVING

- `GROUP BY` groups the selected rows based on one or more attributes
- `HAVING` chooses the grouped rows (by GROUP BY clause) based on a condition

```
select
    customer_id,
    staff_id,
    COUNT(*),
    SUM(amount)
from
    payment
group by
    customer_id,
    staff_id;
```

```
-- What is the minimal length (>46) for films with different rating ?
select
    rating,
    MIN(length)
from
    film
```

```
group by
    rating
having
    MIN(length) >46;
```

ORDER BY

`ORDER BY` orders the selected rows based on one or more attributes. Can be followed by `ASC` or `DESC`

```
select
    *
from
    actor
order by
    first_name,
    last_name,
    actor_id;
```

Relational Set Operations

Join Tables

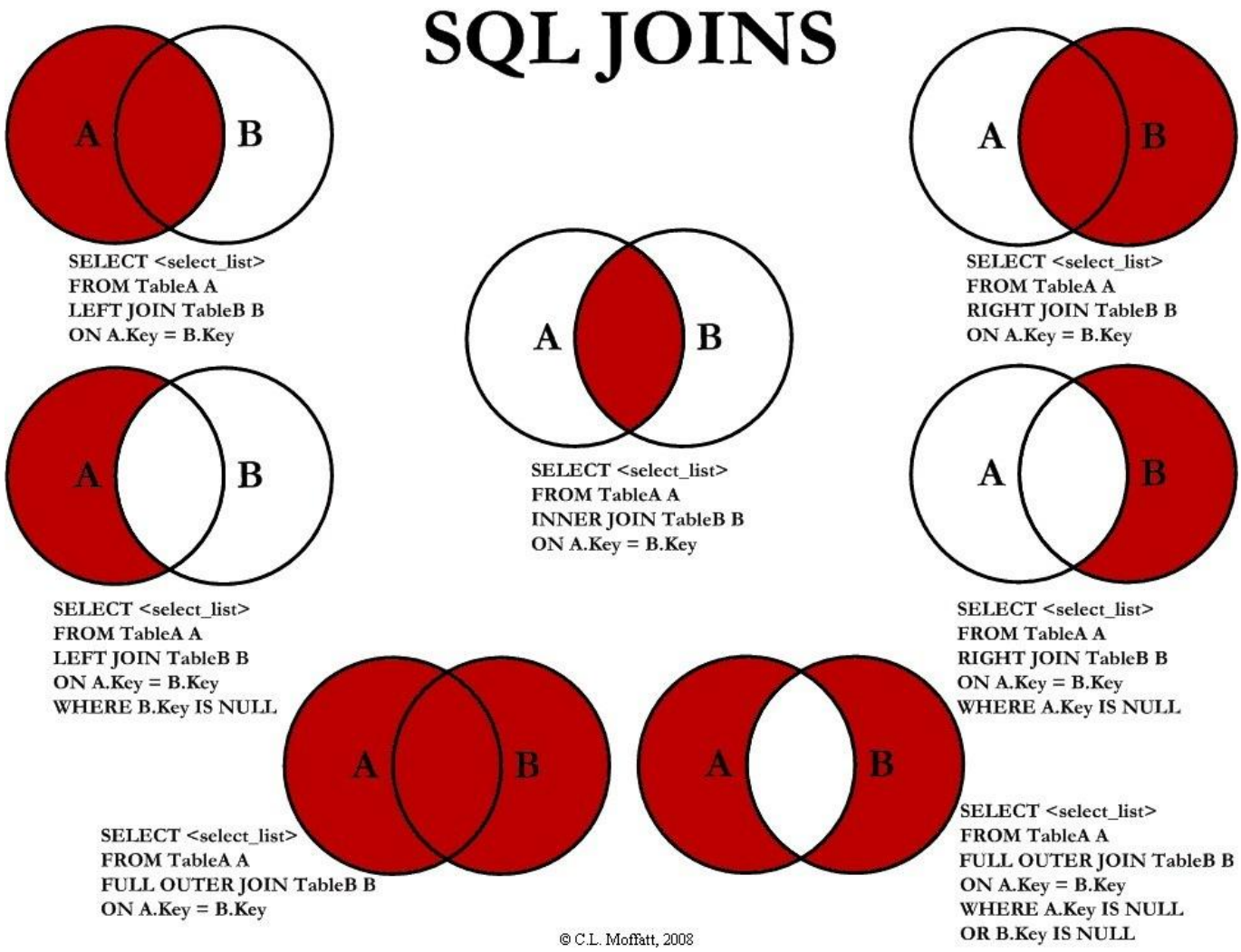
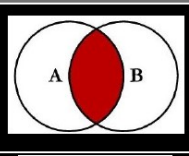
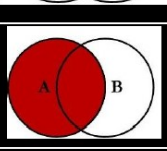
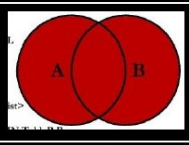
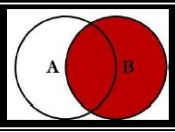


Table	Join Type		Table	Statement	What we use	Visualization
A		Inner	B	A inner join B	A Inner Join B	
	Left	Outer		A Left outer join B	A Left Join B	
	Full			A Full outer Join B	A Full Join B	
	Right			A Right Outer Join B	A Right Join B	

In MySQL, we use `UNION` instead of `UNION`, `INTERSECT`, and `EXCEPT`. `UNION` is set strict and no duplicates; if you want to keep duplicates, you can use `UNION ALL`.

```
-- find out actor whose first name is 'Joe' and customer whoes first name is
'Lisa'.
select
```

```
    first_name,
    last_name
from
    actor
where
    first_name = "Joe"
union
select
    first_name,
    last_name
from
    customer
where
    first_name = "Lisa";
```

```
-- inner join table city and country with country_id.
select
    *
from
    city
inner join country on
    city.country_id = country.country_id;
```

Difference between `IN` and `ON`: `IN` is followed by a table while `ON` is followed by a conditional statement.

Datatypes

- Character and text
  - `CHAR(fixed_len)` len<=255, fixed-length character strings. Strings shorter than expected will be padded with spaces to reach the fixed length.
  - `VARCHAR(max_len)` Variable-length character strings.
  - `TEXT` Character large object (other DMBSs "CLOB").
  - `BLOB` Binary large object. Can store images, sounds, videos, PDF files, Word files, etc.
- Numeric
  - `INT`/`INTEGER` from -2,147,483,648 to 2,147,483,647.
  - `BOOL`/`BOOLEAN` 0 (false) or 1 (true)
  - `DECIMAL(m,d)` fixed-point, fixed number of digits; m is total digits (1~65), d is digits right of the decimal (0~30)
  - `FLOAT`/`DOUBLE` floating-point, up to 7/15 significant digits, less precise than DECIMAL but can store larger/smaller values
  - `UNSIGNED` only +, doubles the maximum of the datatype, e.g. `UNSIGNED INT` `UNSIGNED DECIMAL`
- Date and time
  - `DATE` yyyy-mm-dd, from 1000-1-1 through 9999-12-31
  - `TIME` hh:mm:ss, from -838:59:59 through 838:59:59
  - `DATETIME` Combination of `DATE` + `TIME`, yyyy-mm-dd hh:mm:ss, from 1970-1-1 to 9999-12-31

- `TIMESTAMP` Similar to `DATETIME`, but from 1970-1-1 to 2037-12-31. Can automatically change date by user time zone
- `YEAR[(4)]` e.g. "2021", "2000"

`TIMESTAMP` and `DATETIME` can keep track of when a row was inserted or last updated

Data type conversion

```
cast(expression as cast_ type)
```

Built-in Functions

Strings

```
select
    concat('a', space(10), 'b'),
    -- 'a          b'
    length('very good'),
    -- 9
    upper('very good'),
    -- 'VERY GOOD'
    lower('VERY GOOD'),
    -- 'very good'
    reverse('very good'),
    -- 'doog yrev'
```

Numeric

```
select
    round(-3.14),
    -- -3
    ceiling(-3.14),
    -- -3
    floor(-3.14),
    -- -4
    abs(-3.14),
    -- 3.14
    sign(-3.14),
    -- -1
    rand(),
    -- a random float number, [0.0, 1.0)
    rand(),
    -- another random float number
    rand(123),
    -- rand() with seed
```

```
    power(3.14, 3),
    -- 30.959144
```

Date and Time

```
set @t = "2021-11-28 20:23:51"; -- setting a variable

select
    current_date(),
    current_time(),
    current_timestamp(),
    utc_date(),
    utc_time(),
    year(@t),
    month(@t),
    dayofmonth(@t),
    dayofweek(@t);

select
    extract(year from @t),
    extract(day_second from @t);

select
    date_add(@t, interval 1 month),
    date_sub(@t, interval 1 day);

select
    datediff("2021-11-21", "2021-11-1"),
    -- returns the number of days
    to_days("2021-11-21"),
    -- returns the number of days since the year 0. Not reliable for dates <1582
    time_to_sec("0:10");
    -- returns the number of seconds since midnight 00:00
```

if()

```
select
    title,
    rating,
    if(rating != "R",
    "good film",
    "x") as good_movie
from
    film
```

Create and Delete Databases

```
create database coursedb;
create table Stu(
    stu_id int not null,
    name varchar(30) default "Not available",
    primary key (stu_id));
drop table Stu;
drop database coursedb;
```

Column constraints

- `NOT NULL` NULL values not allowed
- `UNIQUE` no duplicates
- `AUTO INCREMENT` e.g. for an integer column, each new insertion would add 1 to it
- `DEFAULT default_value` convenient to have a default value
- `CHECK(P)` check the data value being entered into a record
- `PRIMARY KEY` A valid relation (table) should have a primary key. By default, `PRIMARY KEY` == `NOT NULL` + `UNIQUE`
- `FOREIGN KEY` A field (or collection of fields) that refers to the `PRIMARY KEY` in another table

```
create table table_name(
    column_name_1  type  column_constraints,
    column_name_2  type  column_constraints,
    …,
    [constraint name] primary key (column_name_1),
    [constraint name] foreign key (column_name_2) references table_name_2
(column_name_1),
    check(column_name_2>2))
```

Referential Integrity ensures that a value that appears in one relation for a given set of attributes also appears for a certain set of attributes in another relation.

Functions and Procedures

Function

return values and can be run like built-in functions

```
delimiter // -- change default delimiter from ";" to "//"
create function my_add(x integer, y integer)
returns integer
deterministic -- state same result on same input, not essential
    begin
        return x + y;
    end //
delimiter ; -- new delimiter "//"
```

```
select my_add(1,2); -- test
```

Procedure

do not return values (`IN`/`OUT`/`INOUT` parameters) and can be run using `CALL` keyword

```
delimiter //
create procedure get_film()
deterministic
begin
    select
        film_id,
        title
    from
        film;
end//
delimiter ;

call get_film();
```

> Warning: Use "Execute script" instead of "Execute SQL statement" in DBeaver to avoid error.

IF THEN ELSE

```
delimiter //
create procedure my_compare(a int, b int)
deterministic
begin
    if a > b then
        select
            "a is larger than b";
    elseif a = b then
        select
            "a equals b";
    else
        select
            "a is smaller than b";
    end if;
end//
delimiter ;

call my_compare(10, 20);
```

Simple LOOP

```
delimiter //
create procedure cumsum(N int)
deterministic
begin
    declare s int default 0;
    declare i int default 1;
    my_loop: loop
        set s = s + i;
        select
            i as added,
            s as result;
        set i = i + 1;
        if i>N then
            leave my_loop;
        end if;
    end loop;
end //
delimiter ;

call cumsum(100);
```

Database Design

Database requirements are statements that define the details and constraints of the data and metadata, which can be represented in a ***conceptual database model***, such as ***Entity-Relationship (ER) model***. The result of ER modeling is an ***ER diagram (ERD)***.

Entity-Relationship Diagram (ERD)

We use a modified version of ***Peter Chen's Notation***. View full info

Database Normalization

Normalization is a process to improve the design of relational databases, mainly to reduce data redundancy while preserving information. Normal form (NF) is a set of particular conditions upon table structures.

0NF -> 1NF

> 1. Remove duplicated rows.
> 2. Eliminate multivalued columns.

A table is in 1NF if each row is unique and not duplicated. Within each row, each value in each column must be single valued.

1NF -> 2NF

> Create an additional relation for each set of partial dependencies.

> 1. The portion of the primary key in partial dependency => primary key of the new table (becomes a foreign key in original table).
> 2. The columns determined in partial dependency => columns of the new table (removed from original table).
>
> The original table remains after the process of normalizing to 2NF, but no longer contains the partially dependent columns.

A table is in 2NF if it is in 1NF and does not contain ***partial dependencies*** (a column of a relation is functionally dependent on a portion of a composite primary key). Table has a single-column primary key <=> Table is in 2NF.

2NF -> 3NF

> Create additional relations for each set of transitive dependencies in a relation.
>
> 1. The transitively determinant nonkey column in the original table => the primary key of a new table.
> 2. Move the determined nonkey columns to the new table.
>
> The original table remains after normalizing to 3NF, but it no longer contains the transitively dependent columns

A table is in 2NF if it is in 2NF and does not contain ***transitive functional dependencies*** (nonkey columns functionally determine other nonkey columns).

Denormalization

Denormalization is reversing the effect of normalization by joining normalized relations into a relation that is not normalized in order to improve query performance.

Database Indexing

Indexing is a mechanism for increasing the speed of data search and data retrieval on relations with a large number of records.

- ***Linear Search*** O(n) in the worst case not indexed

- ***Binary Search*** O(log(n)) creates an additional index table (sorted + pointer), and allows binary search on it and then points back to the original column (unsorted) to increase the speed of search and retrieval on the columns that are being indexed.

CUSTOMER

CustID	CustName	Zip
1000	Zach	60111
1001	Ana	60333
1002	Matt	60222
1003	Lara	60555
1004	Pam	60444
1005	Sally	60555
1006	Bob	60333
1007	Adam	60555
1008	Steve	60222
1009	Pam	60333
1010	Ema	60111
1011	Peter	60666
1012	Fiona	60444

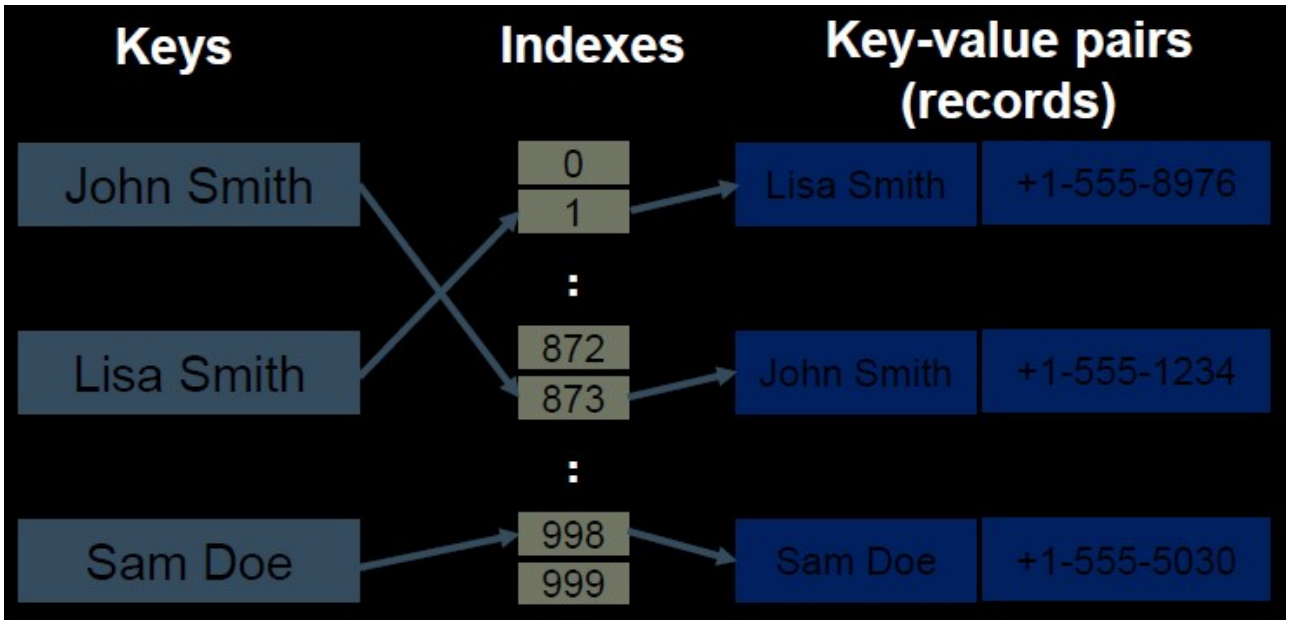
Search on sorted column is faster than on unsorted column!
Because during binary partitions:
n -> n/2 -> n/4 -> n/8 -> … -> 1
Let partition number be k,
then n/(2^k) ~ 1. Get k ~ log(n).

→ Step 1 Eliminate records from here, above (since CustID value is lower than 1008)

→ Step 3 Customer 1008 found

→ Step 2 Eliminate records from here, below (since CustID value is higher than 1008)

- ***Hash Index*** O(1) is useful when files are not sequential. Data in the system distributes to storage spaces called buckets depending on the key value calculated by hash function.



  - Very efficient at equality queries (column==value).
  - Take constant time, independent of the number of rows in a table.
  - Not efficient at inequality queries (eg. <, <=, >, <=, between, in, not in, like).
  - Require more space than range indexes.

- ***B-tree Index*** O(log(n)) B-tree generalizes the binary search tree to allow more than 2 branches in the nodes. The index tree is stored separately from the data. The lower-level leaves contain the pointers to the actual data rows. More info about B-tree

  - Range indexes are efficient at processing inequality queries (eg. <, <=, >, <=, between, in, not in, like).

- Not fast for equality queries.

```
create index index_name
[using {btree | hash}]
on table_name (column_name [asc | desc],
...);
```

Database Transactions

Database operations have two types:

- ***Read(X)*** query
- ***Write(X)*** insertion, deletion, update

Database transactions have two types:

- ***Read-only*** only read(X) operations
- ***Read-Write*** involves write(X) operations

```
begin;
update
    accounts
set
    balance = balance - 100.00
where
    name = 'Bob';
update
    accounts
set
    balance = balance + 100.00
where
    name = 'Alice';
commit;
```

Consistent Database State all data integrity constraints satisfied

- ***integrity rules*** e.g. primary key uniqueness, foreign key references, transaction completeness (enforced by DBMS)
- ***business rules*** e.g. sum of account balance remains 0 after inter-account money transfer (enforced by programmer)

A transaction must begin with a consistent database state, and end with another consistent state. But the intermediate state during a transaction could be inconsistent.

Transaction Requirements – ACID

- ***Consistency*** On database state, a transaction can only bring it from one consistent state to another by preventing data corruption.
- ***Atomicity*** A transaction's operations should be executed as a single "unit" altogether (all-or-none).
- ***Durability*** Results of a successful transaction are permanently stored in the system (even in case of power loss or system failures).

> The above is about a single transaction, called CAD requirements. In reality, multiple transactions can occur at the same time and access the same data items. Thus, we have ACID requirements for multiple transactions.

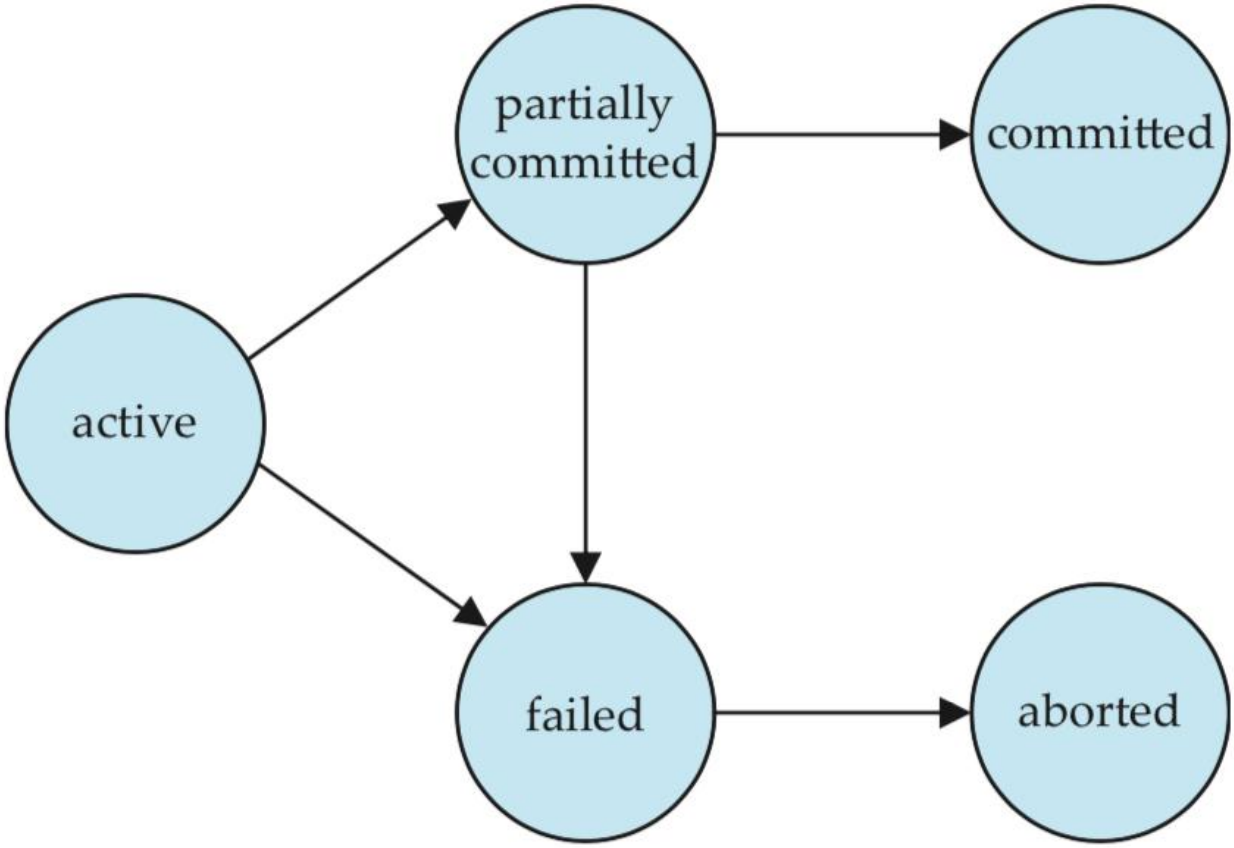
- ***Isolation*** Transactions are executed independently/isolated from each other. Intermediate results of a transaction is not visible to others.

Transaction Atomicity and Durability

- ***Atomicity*** Transaction in "all-or-none" execution mode
  - ***committed*** Transaction initiates and can complete its execution successfully
  - ***aborted*** Transaction fails at somewhere and can not complete successfully. It then proforms a rollback, all its executions undone
- ***Durability*** Once a transaction has been committed, the effects can not be undone by aborting it

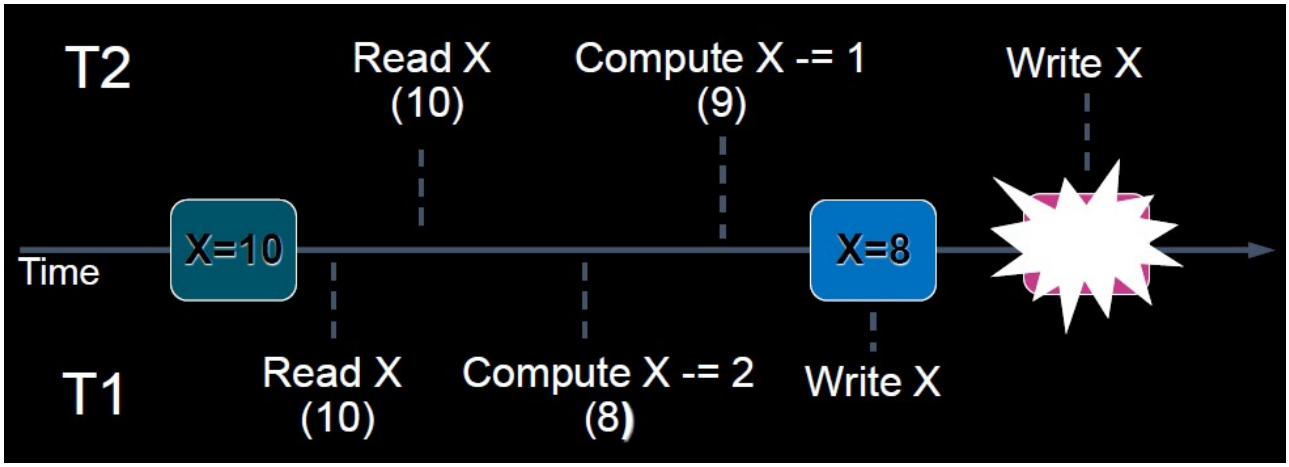
Transaction States

- ***Active*** the initial state; the transaction begins from here.
- ***Partially Committed*** after the final statement has been executed.
- ***Failed*** after the discovery that normal execution can not proceed.
- ***Aborted*** the transaction rolled back and restored to previous state.
- ***Committed*** after successful completion.



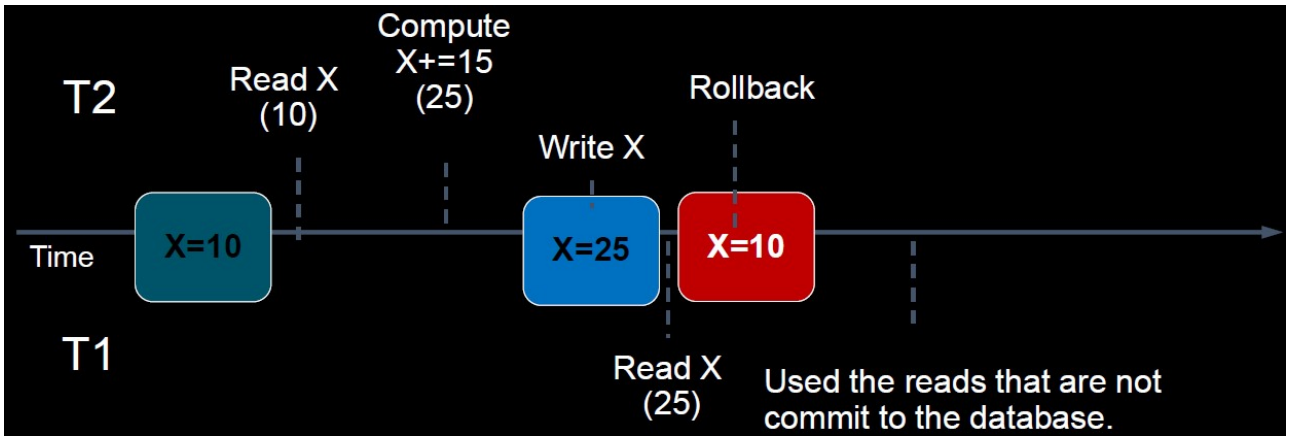
Concurrent Transaction Common Problems

- ***Lost Update*** ("modified after write", write-write)
  two concurrent transactions update the same data element, and one of the updates is lost (overwritten by the other transaction)



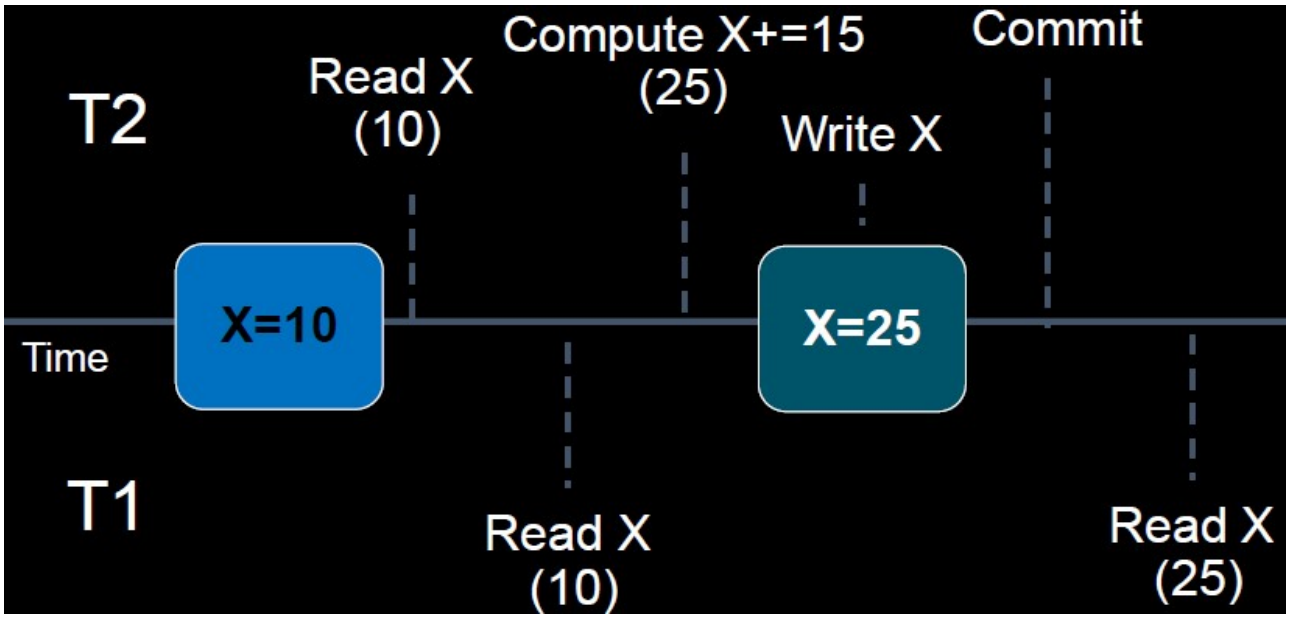
  T1 and T2 read the same data and update the data concurrently. The results submited by T2 cause the lost of update by T1.

- ***Dirty Read*** ("modified before read", write-read-rollback)
  a transaction reads data from a row that has been modified by another running transaction (but not yet committed)



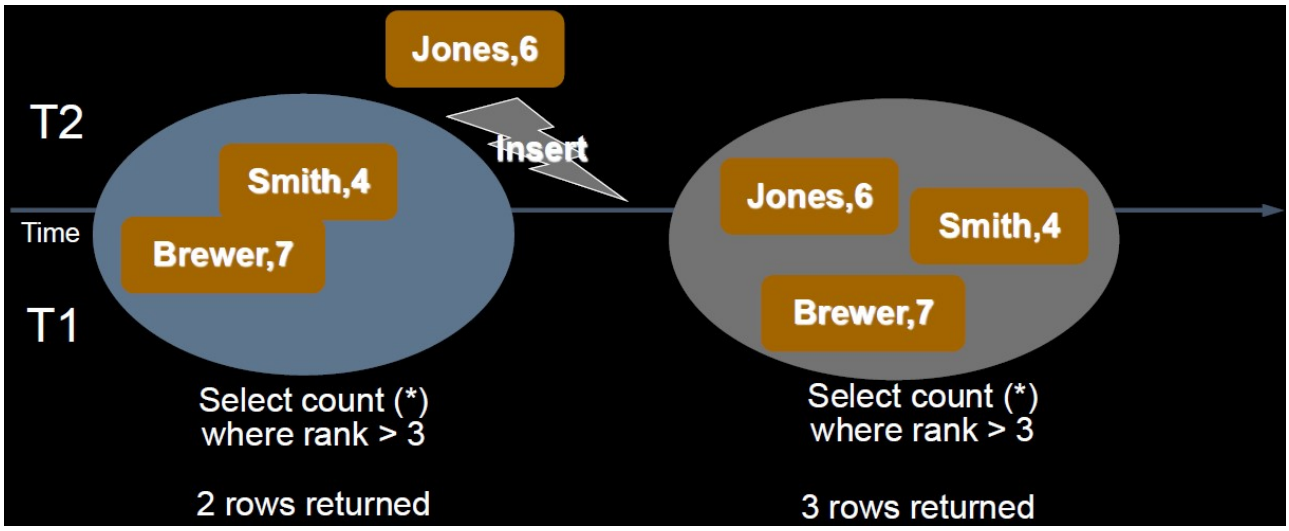
Just before T1 reads some data, T2 updates the same data. However, after that, T2 performs a rollback due to some reason. Now the data read by T1 is inconsistent with the data in the database.

- ***Non-repeatable Read*** ("modified between two reads", read-write-read)
  a transaction reads the same data element twice, but the data element is changed by another transaction between the two reads



T1 reads some data, T2 then updates the data, so when T1 reas the data again, the data is inconsistent with previous ones.

- ***Phantom Read*** ("modified after read", read-write)
  a transaction queries the table, but new rows are added or removed by another transaction to the records being read



T1 reads some data based on some conditions, then T2 inserts some new data that matches the condition. (if T1 searches for data with the same condition, more records are returned)

Transaction Isolation Levels

- **Serializable** the most restrictive one.
- ***Repeatable Read*** allows only committed data to be read and further requires that, between two reads of a data item by a transaction, no other transaction is allowed to update it.
- ***Read Committed*** allows only committed data to be read, but does not require repeatable reads. For instance, between two reads of a data item by the transaction, another transaction may have updated the data item and committed.
- ***Read Uncommitted*** allows uncommitted data to be read. It is the lowest isolation level allowed by SQL.

Isolation Level	Dirty Read	Lost Update	Non-repeatable Read	Phantom Read
Serializable	N	N	N	N
Repeatable Read	N	N	N	Y
Read Committed	N	Y	Y	Y
Read Uncommitted	Y	Y	Y	Y

Serializable Schedule

- ***Serializable Schedule*** Interleaved execution of transactions that are equivalent to some serial schedule of these transactions
- ***Equivalent Schedules*** Two schedules that yield the same results on the same transactions

> If two operations on the same data has at least one "write" in them, then they can not swap, otherwise will cause a conflict (loss of equivalence) before and after swapping.
>
> 1. I = read(Q), J = read(Q);
> 2. I = read(Q), J = write(Q);
> 3. I = write(Q), J = read(Q);
> 4. I = write(Q), J = write(Q);

> Case 1 to case 4 have different results if we swap the order, so I and J operations are conflict in above case 2~4.

- ***(Conflict) Equivalence*** means a set of schedules can be transformed into each other by a series of swaps of nonconflicting instructions.
- ***(Conflict) Serializability*** means a schedule is (conflict) equivalent to a serial schedule.

Further explaintion in Chinese

Transaction management

- ***Autocommit Mode*** default mode in MySQL
  - Each statement is a transaction, as if it were surrounded by `BEGIN` and `COMMIT`
  - If an error occurs during statement execution, the statement is automatically rolled back (can not control it using `ROLLBACK`)
- ***Transaction Mode*** use `BEGIN` and `COMMIT` to switch to transaction mode
  - Supports multiple statements/operations
  - With `BEGIN`, autocommit is disabled until the transaction ends with `COMMIT` or `ROLLBACK`

Practice

- to dump a database named dvdrental

```
cd "C:\Program Files\MySQL\MySQL Server 8.0\bin"
mysqldump -u root -p --databases dvdrental > my_database.sql
```