Lecture 05: SQL DML and TCL

1. ***SQL DML (Data Manipulation Language)***
2. ***Insert data***
3. ***Copy table***
4. ***Update data***
5. ***Default value***
6. ***Manage date***
7. ***Delete data***
8. ***SQL TCL (Transaction Control Language)***
9. ***Concurrency***
10. ***Database transaction***
11. ***Commit and rollback***
12. ***Sessions***
13. ***ACID properties***
14. ***Locking***

Pär Douhan, pdo@du.se

SQL - DML

Let's start with SQL - **DML**

SQL DML

The abbrivation DML stands for Data Manipulation Language

Used when inserting, updating or deleting data from a table. DML will always **change** the **data content** of a table/tables.

TABLE

column1	column2	column3

1. Add data: **insert**
2. Change data: **update**
3. Delete data: **delete**

The table contains data in the form of rows. It is these rows we **insert**, **update** and **delete** with **DML**.

Insert data

When we **load a database table with data**, we can do it in slightly different ways. Here are some examples:

1. Add new rows with **insert.**
2. Load a table with data **from another table**.
3. Load one or more tables with data from an **external file**.

Data capture

The web browser is a common interface for adding data in a database.

Often, data entered into the database comes from different types of web forms. This happens if you, for example. place an ad on eBay or make a visit to your internet bank and pay bills or you might answer a web survey.

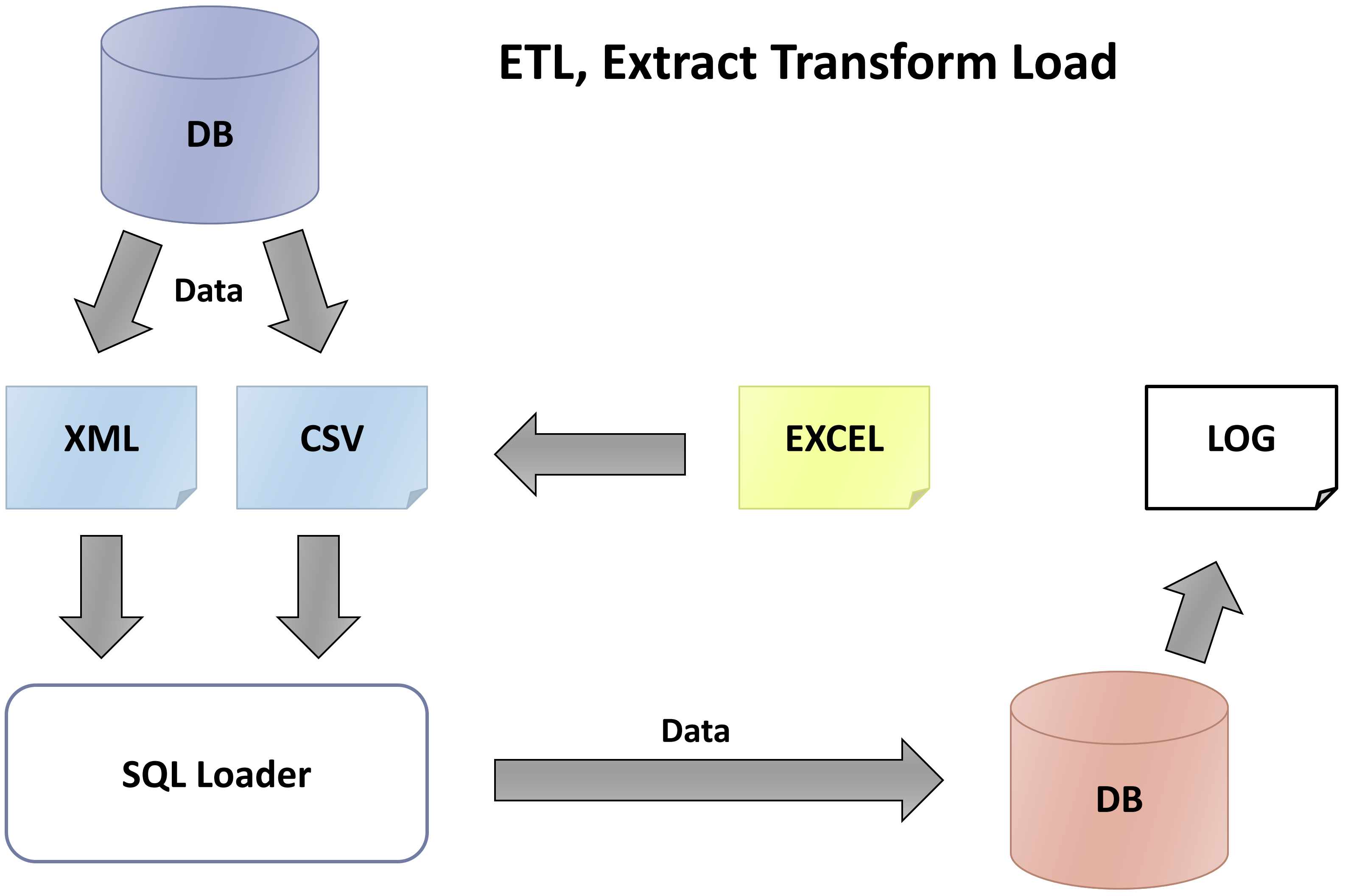
The content of your web form is sent to the web server which, through various drivers ODBC, OLEDB or maybe JDBC together with Node.js, PHP, .NET or JSP, connects to the database and executes an SQL statement which, in this example, insert the ad's data into the database.

In other cases, data may come from other databases in the form of JSON, XML files or plain text files (CSV files). The data in these files can be loaded into the database with **various tools**, e.g. Oracle SQL Loader.

Excel files are another common source of data. The data content in excel files can also be loaded into a database, often the contents of the excel file are exported to a so-called csv file, which is a plain text file, before loading data into the database.

SQL Loader

ETL, Extract Transform Load



Add data with insert

```
CUSTOMER
# cnr
* persnr
* fname
* ename
* regdate
```

CUSTOMER

cnr	persnr	fname	ename	regdate
1	840315-7415	Erik	Ek	20191107
2	870201-5510	arman	ikovic	20191107
3	745022	karin	ander	20191107

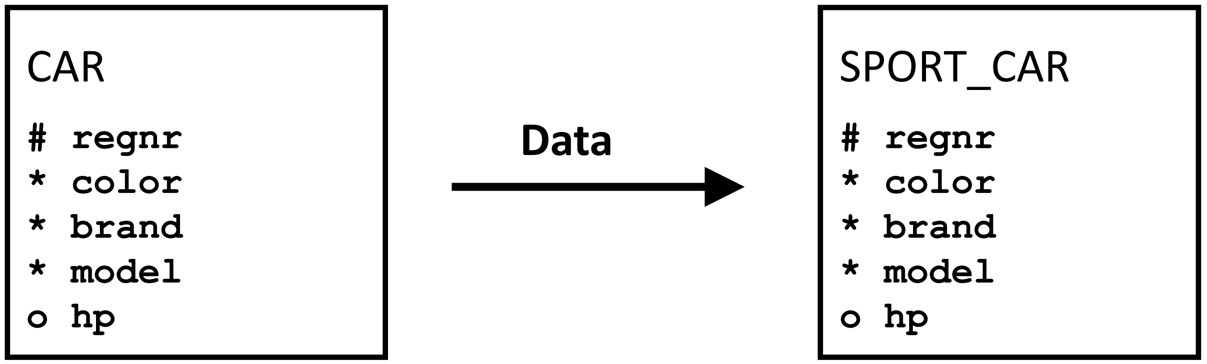
```
insert into customer(cnr,persnr,fname,ename,regdate)
values(seq_cnr.nextval,'840315-7415','Erik','Ek',sysdate);

insert into customer(cnr,persnr,fname,ename,regdate)
values(seq_cnr.nextval,'870201-5510','arman','ikovic',sysdate);

insert into customer(cnr,persnr,fname,ename,regdate)
values(seq_cnr.nextval,750610-5588,'karin','ander',sysdate);
```

If we forget single quotation marks ' around persnr, Oracle will take 750610 minus 5588 and type convert the difference to varchar2

Load data from another table



```
CAR
# regnr
* color
* brand
* model
o hp
```

```
SPORT_CAR
# regnr
* color
* brand
* model
o hp
```

Task: Fill the sports car table with data from the car table.

```sql
insert into sport_car(regnr,color,brand,model,hp)
select regnr,color,brand,model,hp
from car;
```

Or like this (* stands for all columns):

```sql
insert into sport_car
select *
from car;
```

Copy a table with or without data

```
CAR

# regnr
* color
* brand
* model
o hp
```

```
FAST_CAR

# regnr
* color
* brand
* model
o hp
```

Task: Create the Fast Cars table as a copy of the Cars table. Only bring in car items that have **more than 400 hp**.

```
create table fast_car as
select *
from car
where hp > 400;
```

The Sports Car table is created as a copy of the Car table. Data and NOT NULL constraints are included.

What is **not** included is :

PK, **FK**, **Check** and **Unique**

```
create table car2 as
select *
from car
where 3 = 4;
```

The Car2 table is created as a copy of the Car table. **NOTE!** No data is included. This then the condition: 3 = 4 never becomes true**.**

Update a single row

```
CUSTOMER
# cnr
* persnr
* fname
* ename
* regdate
```

KUND

cnr	persnr	fname	ename	regdate
1	840315-7415	Erik	Ek	20191107
2	870201-5510	arman	ikovic	20191107
3	750610-5588	karin	ander	20191107

Task: Correct the incorrect social security number from previous examples. We do this by updating the cell that contains persnr 745022 for the customer with cnr = 3.

```sql
update customer
set persnr = '750610-5588'
where cnr = 3;
```

This is called a *singleton update*. This is because there is only one row that is affected.

Activate default value

```
CUSTOMER
# cnr
* persnr
* fname
* ename
* regdate
```

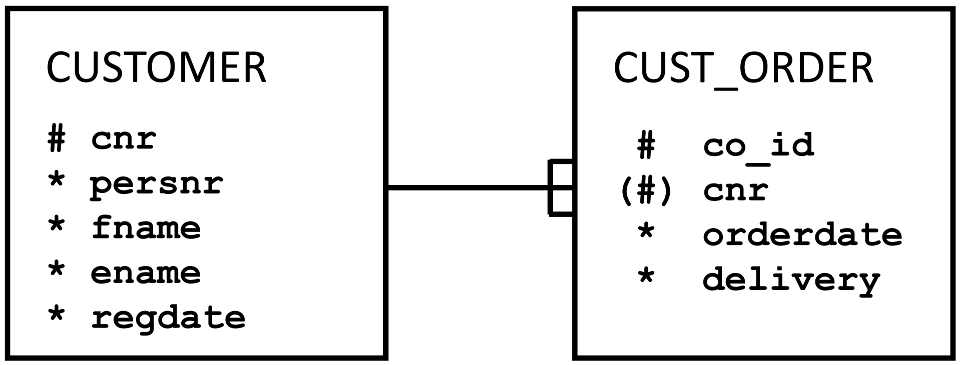
How do we do to **activate the default value** of the column regdate?

```
Declared as: regdate date default sysdate not null,
```

If we **omit** the column regdate in our insert statement, then Oracle will trigger the default value for the column.

```
insert into customer(cnr,persnr,fname,ename)
values(seq_cnr.nextval,'921111-5542','Rolf','Ek');
```

Insert date



We add a row to the Customer Order table. Order date is sysdate and delivery date will be in 4 days. **How do we do?**

```
insert into cust_order(co_id,cnr,orderdate,delivery)
values(seq.nextval,'2559',sysdate,sysdate + 4);
```

date + number = date

Insert date

Study!

```
insert into cust_order(co_id,cnr,orderdate,delivery)
values(seq.nextval,3,sysdate,to_date('2018-10-28','YYYY-MM-DD'));
```

```
orderdate = 2018-10-24:14:30:01

delivery = 2018-10-28:00:00:00
```

Why?

```
insert into cust_order(co_id,cnr,orderdate,delivery)
values(seq.nextval,3,sysdate,to_date('2018-10-28:14:10','YYYY-MM-DD:HH24:MI'));
```

We have to set the "*fine grain*" ourselves using the *desired format mask* for the date string.

Update several rows

```
PRODUCT

# prodnr
* prodname
* stock
* price
```

It will be a great Black friday sale. We will therefore reduce the price of all our items by 30%.

```
update product
set price = price – (price * 0.3);
```

If we do **not** have any where condition, then **all rows** will be affected.

```
update product
set price = price – (price * 0.3);
where stock > 100;
```

If we have a where condition that does **not include a primary key or other column that is unique**, then we can affect one or more rows with our SQL statement.

Update several columns

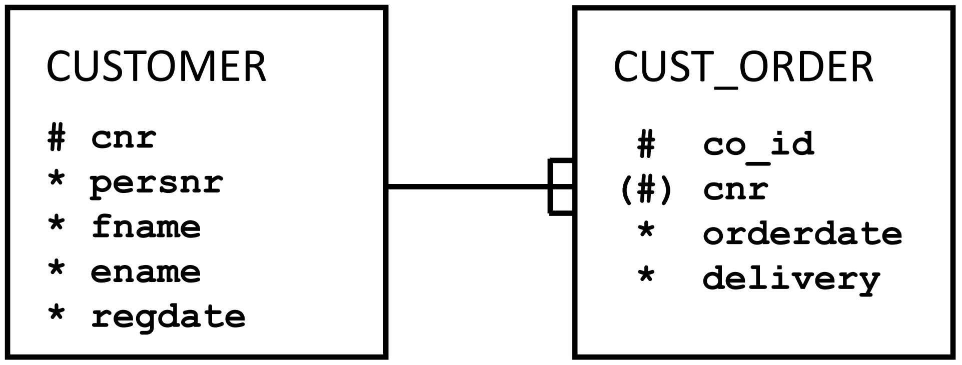
```
CUSTOMER
# cnr
* persnr
* fname
* ename
* adress
* zip
* city
* email
* regdate
```

A customer logs into his account to update his customer information with his new address.

The following SQL statement must be executed for the data to be changed:

```
update customer
set adress = 'Hemgatan 12', zip = 84523, city = 'Mora'
where cnr = 12568;
```

Delete one row



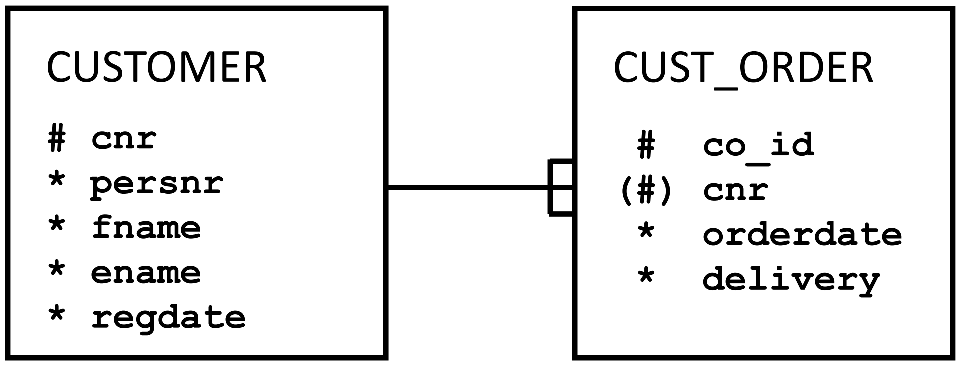
We want to remove customer with **cnr** = 3 from the customer table :

```
delete
from customer
where cnr = 3;
```

```
Error at line 1:
ORA-02292: integrity constraint (TEST.CUST_ORDER_CNR_FK) violated - child Records Found.
```

This means you tried to delete a row in a parent table where a child row still existed.

Emptying the tables on data



If we want to clear both tables on the data, we do the following:

```
delete
from cust_order;

delete
from customer;

commit;
```

Clear the tables in the correct order. End all successful transactions with **commit**.

Lab Report

How do I present the lab?

Readable code is important!

```
create table KUND (
 persnr varchar2(11),
 username varchar2(12) not null,
 passwd varchar2(12) not null,
 fnamn varchar2(40) not null,
 enamn varchar2(60) not null,
 kredittyp varchar2 (12) not null,
 telnr varchar2(14)
 );
```

Which code is most readable?

Should be this one

create table KUND (
 persnr varchar2(11),
 username varchar2(12) not null,
 passwd varchar2(12) not null,
 fnamn varchar2(40) not null,
 enamn varchar2(60) not null,
 kredittyp varchar2 (12) not null,
 telnr varchar2(14)
 );

```
create table KUND (
 persnr varchar2(11),
 username varchar2(12) not null,
 passwd varchar2(12) not null,
 fnamn varchar2(40) not null,
 enamn varchar2(60) not null,
 kredittyp varchar2 (12) not null,
 telnr varchar2(14)
);
```

Group what belongs together

Write a well-structured, clear and easy-to-read lab report!

```
Task x: Create tables

-- Table CUSTOMER --
create table customer(
cnr number(6),
fname varchar2(50) not null,
credit_type varchar2(5));


alter table customer
add constraint customer_cnr_pk primary key(cnr)
add constraint customer_credit_type_ck check(kredittyp in('high','average','low'));


-- Table CUST_ORDER --
 Definitions


-- Tabell x --
 Definitions
```

Syntax highlightning

Online syntax highlighting for the masses! for "SQL, PL/SQL"

https://tohtml.com

SQL - TCL

We continue with SQL - **TCL**

SQL TCL

The abbrivation TCL stands for Transaction Control Language

Used when we want to ***control*** the effects of a ***transaction*** (**T**). A **T** is created by **DML**, i.e. when we **insert**, **update** or **delete** against a table/tables.

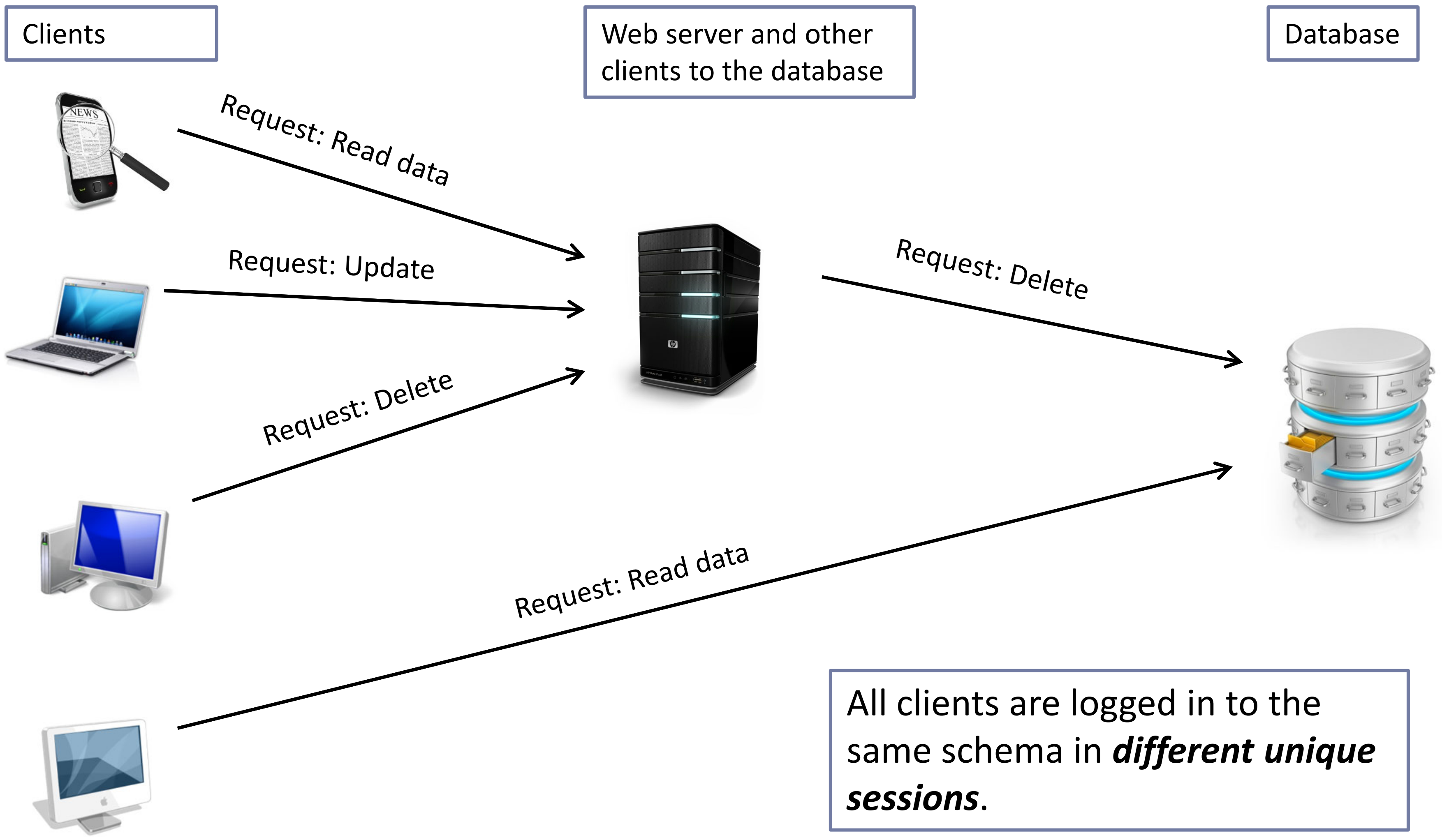
TABLE

column1	**column2**	**column3**

1. Save the effects of a T: **commit**
2. Undo the effects of a T: **rollback**

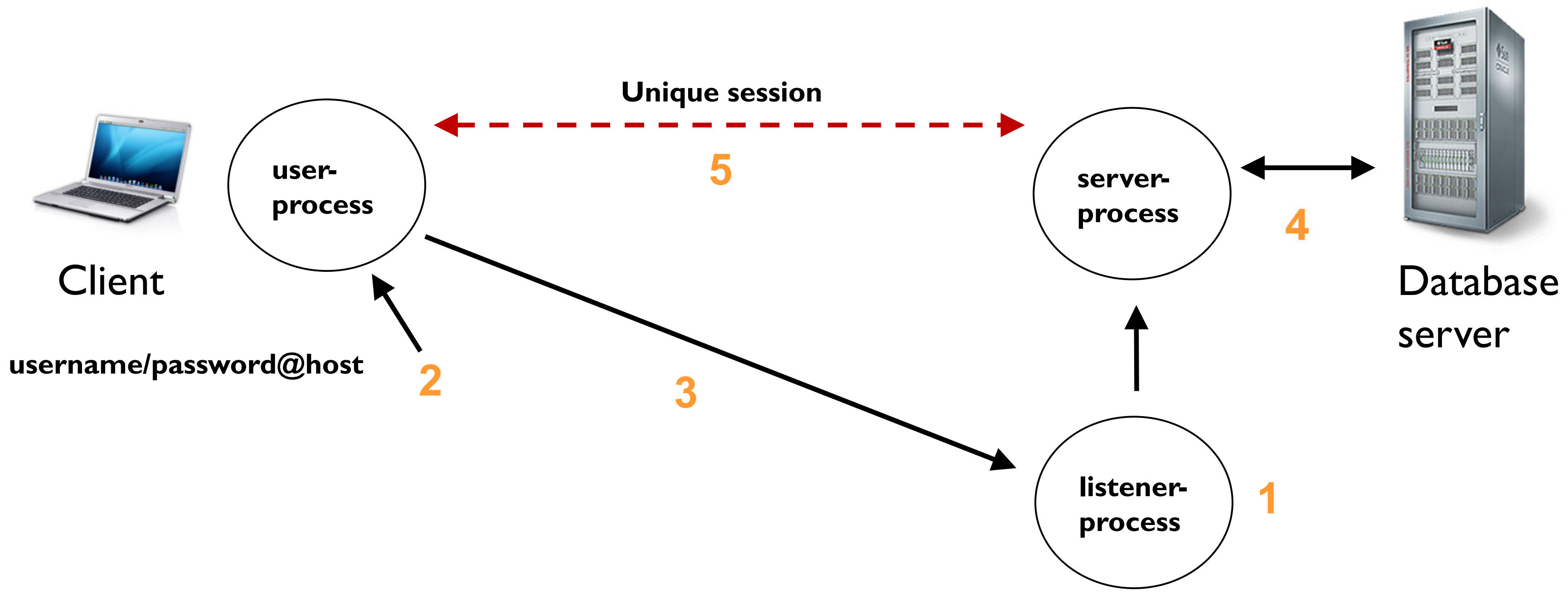
When we affect the data content of a table with DML, a **transaction** is created. A **T** will always **change** the **data content** of a table/tables.

Concurrency



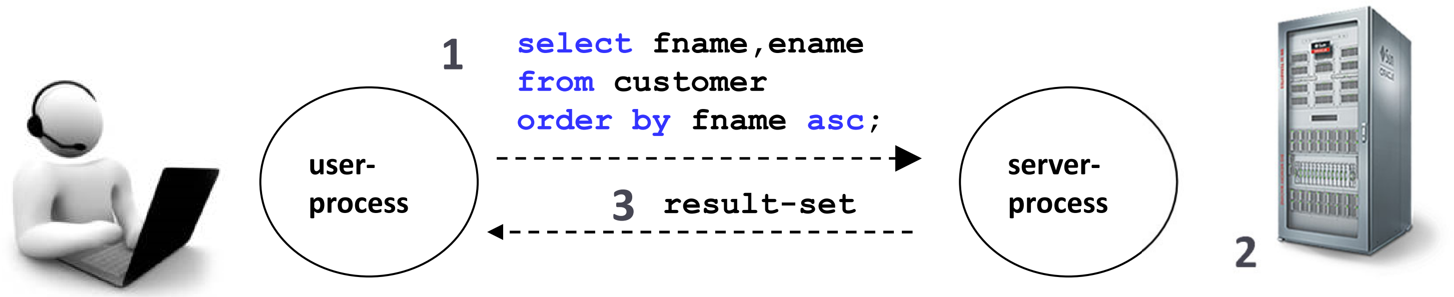
All clients are logged in to the same schema in ***different unique sessions***.

Connect to a database server



1. *Listener-process* listens for calls over the network.
2. The client starts a program and logs in. This creates a *user-process* at the client.
3. The call is noted by *listener-process.*
4. A *server-process* starts on the database server. This *server-process* then communicates with Oracle Instance (RDBMS).
5. *User-process* communicates with the *server-process* for the rest of the *unique session* length.

"Run a question"



```
select fname,ename
from customer
order by fname asc;
```

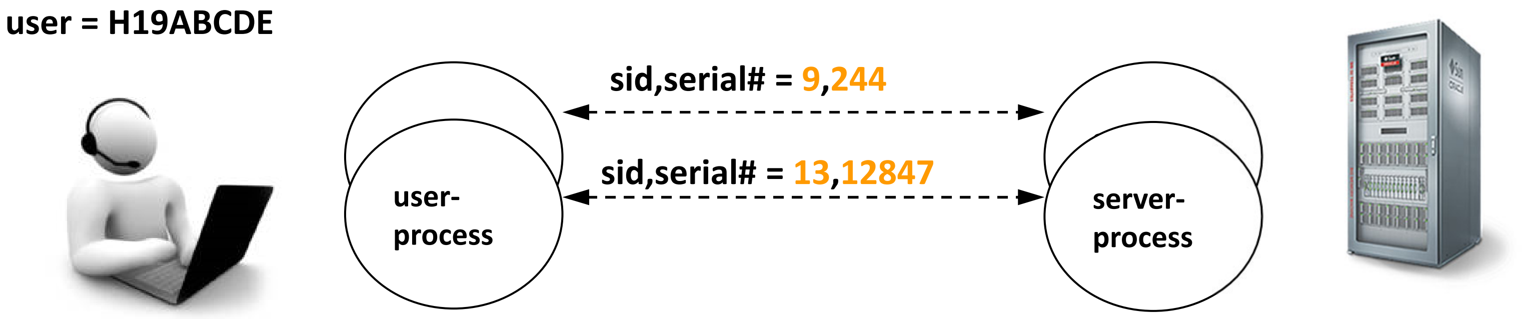
"Run a question" goes in three steps :

1. **PARSE:** *user-process* sends the SQL statement to the *server-process* and requests compilation. Oracle Instance compiles and returns status: success or failure (some kind of error message).

2. **EXECUTE:** Oracle Instance runs a background-task (*db writer*) which collect data from the *data files* or from *data base buffer cache*.

3. **FETCH:** *server-process* returns the result-set from the Customer table, sorted by first name to the *user-process.*

Different sessions



Sometimes a session can "**lock itself**" and must be "**killed**":

```
alter system kill session 'session-id,session-serial'
```

The above command "**kills**" specific session. We find session id and session serial parameters in **v$ session** view (the columns sid and serial #). That is in ***the data dictionary***.

```
alter system kill session '9,2444'
```

```
select sid,serial#
from v$session
where username = 'H19ABCDE';

  SID    SERIAL#
------ ----------
     9       2444
    13      12847
```

ACID-Properties

The basic properties of a database transaction, known as **ACID** properties. ACID is an acronym for the following:

- **Atomicity**
  - All tasks of a transaction are performed or none of them are. There are no partial transactions. For example, if a transaction starts updating 100 rows, but the system fails after 20 updates, then the database rolls back the changes to these 20 rows.
- **Consistency**
  - The transaction takes the database from one consistent state to another consistent state. For example, in a banking transaction that debits a savings account and credits a checking account, a failure must not cause the database to credit only one account, which would lead to inconsistent data.
- **Isolation**
  - The effect of a transaction is not visible to other transactions until the transaction is committed. For example, one user updating the hr.employees table does not see the uncommitted changes to employees made concurrently by another user. Thus, it appears to users as if transactions are executing serially.
- **Durability**
  - Changes made by committed transactions are permanent. After a transaction completes, the database ensures through its recovery mechanisms that changes from the transaction are not lost.

Commit and rollback

Commit and **rollback**

- A transaction involvs one or more SQL DML-statements like **insert**, **update** or **delete**

- **Commit:** Make changes done in **transaction** permanent.
- **Rollback**: **Rollbacks** the state of database to the last **commit** point.

The use of transactions is one of the most important ways that a database management system differs from a file system.

Atomic unit of work (atomicity)

```
update employee
set sal = sal - 300
where empno = 120;

update office
set otype = 'basement'
where empno = 120;

update parking
set ptype = 'no roof'
where empno = 120;
```

time = x

```
commit; or rollback;
```

T = **logical unit of work**.
Lower salary, change office and parking lot. requires three DML statements.

During the time x, which passes before T has been commited or rolled back, the values, before the effects of T, are stored in **undo segments**.

Example

user = u233
session x

```
update customer
set ename = 'vlasic'
where custid = 2;
.
.   } time = x
.
commit;
or
rollback;
```

CUSTOMER TABLE

custid	fname	ename
1	carolyn	smith
2	arman	vlasic
3	karen	andersson

DML-lock (new image)

2	arman	brtvic

undo segment (old image)

DML: update and delete generates undo data.

Locked data is only visible in session x

The locks are released at commit or rollback, and the effects of the transaction are visible for all sessions.

user = u233
session y

Runs the following SQL during the time x:

```
select ename
from customer
where custid = 2;
```

?

Integrity problems

The lost update problem

A lost update occurs when two different transactions are trying to update the same column on the same row within a database at the same time.

T1	time	**T2**	balance
fetch(balance) 100	**1**		100
	2	fetch(balance) 100	100
	3		100
update(balance) 1000	**4**		1000
	5	update(balance) 400	400
	6		400

The balance should have been 100 + 900 + 300 = 1300, **T2** *over writes* **T1**.

Solution = **lock**. **T2** should not be able to fetch the balance before **time = 4**.

Different lock types

Integrity Issues such as the lost update problem can be avoided by a technique called locking.

We assume that the database supports two different types of locks:

1. **X-lock** (exclusive locks = write locks)
2. **S-lock** (shared locks = read locks)

T2 wants

Other T has

	X	S
X	N	N
S	N	Y
	Y	Y

N = no

The End

I want vacation!