Lecture 04: SQL DDL

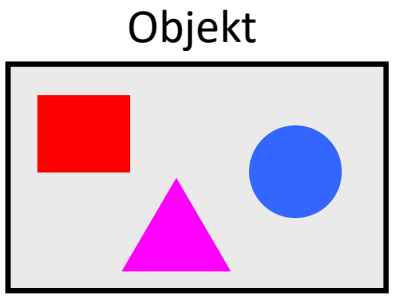
1. *SQL DDL (Data Definition Language)*
2. *Create table*
3. *Add Primary Key*
4. *Data Dictionary Views*
5. *Naming constraints*
6. *Add Foreign Key*
7. *Add a column in a table*
8. *The Sequence object*
9. *The Dual table*
10. *Drop objects*
11. *Data types*
12. *Built-in functions*

Pär Douhan, pdo@du.se

SQL DDL

The abbreviation DDL stands for Data Definition Language

Used when creating, alter or dropping database objects. Some examples of database objects are table, procedure, function, trigger and sequence.



1. Create an object: **create**
2. Alter an object: **alter**
3. Drop an object: **drop**

Create a table

STUDENT

```
# studnr
* first_name
* last_name
o cell
```

STUDENT

studnr	first_name	last_name	cell

```
create table student(
studnr varchar2(10),
first_name varchar2(50) not null,
last_name varchar2(50) not null,
cell varchar2(15));
```

table STUDENT created.

Add a Primary key

STUDENT

```
# studnr
* first_name
* last_name
o cell
```

Now we will add a **primary key** (PK) to the student table. We can see in the model that the column studnr is marked with **#** and should therefore be the table's primary key.

We can do this directly when we create the table, but I prefer to do it afterwards. We use an **alter-statement** to do this:

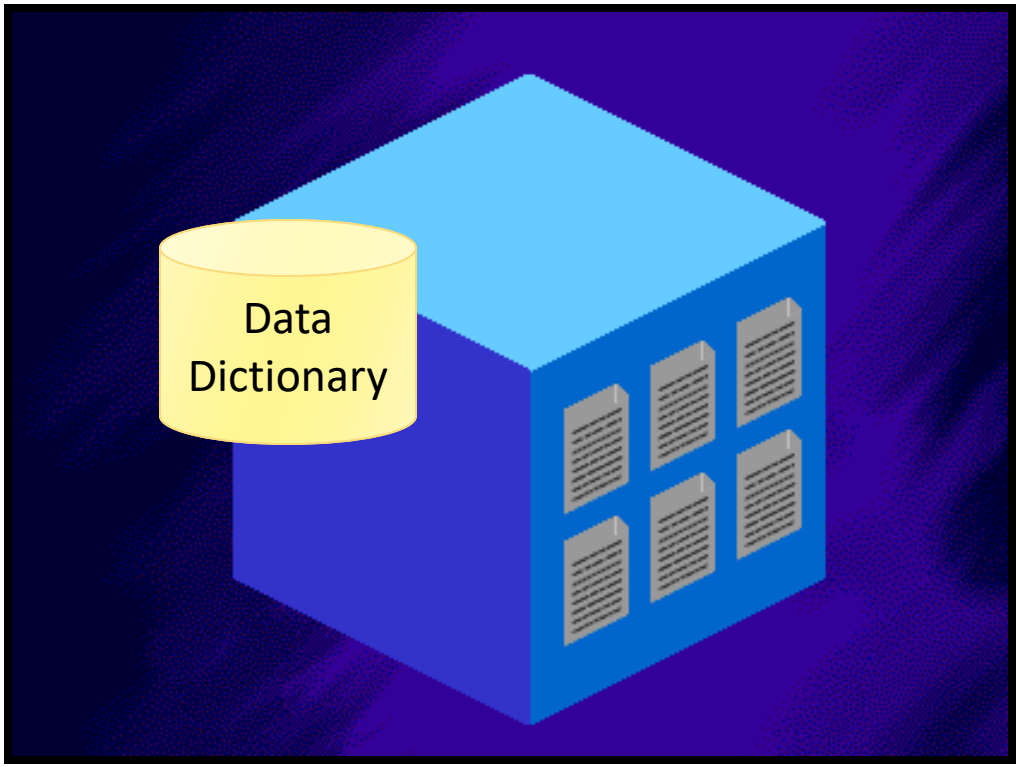
```
alter table student
add primary key(studnr);
```

table STUDENT alterd.

Wondering what name our PK got?

We can find out by searching *Data Dictionary*.

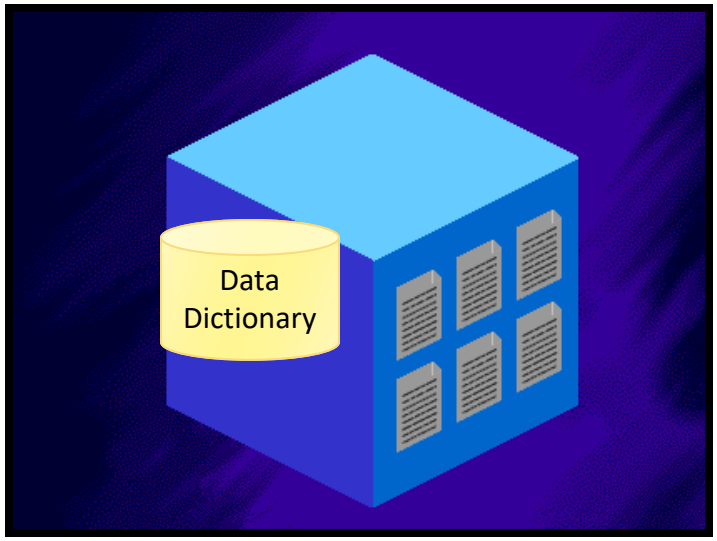
Data Dictionary



A database always contains a description of stored information. This is usually called *Data Dictionary*.

In Oracle, there are several different *data dictionary views*. These are located in a tablespace called *system*.

Data Dictionary Views



Data dictionary views, also known as *catalog views*, allow us to monitor the state of the database in real time:

1. The **USER**, **ALL**, and **DBA** views contain information about schema objects that are available at different access levels.
2. The so-called **V$** views contain information about database performance. For example, we can see how much memory the different sessions of the database consume etc.

Through these views we can find **metadata** about all objects and different states in the database. We find all views and much more on ss64:

http://ss64.com

Search Data Dictionary

STUDENT

 # studnr
*** first_name**
*** last_name**
o cell

```
desc user_constraints;
```

```
Name                 Null Type
-------------------- ---- --------------
OWNER                     VARCHAR2(128)
CONSTRAINT_NAME           VARCHAR2(128)
CONSTRAINT_TYPE           VARCHAR2(1)
TABLE_NAME                VARCHAR2(128)
SEARCH_CONDITION          LONG()
SEARCH_CONDITION_VC       VARCHAR2(4000)
R_OWNER                   VARCHAR2(128)
R_CONSTRAINT_NAME         VARCHAR2(128)
DELETE_RULE               VARCHAR2(9)
STATUS                    VARCHAR2(8)
DEFERRABLE                VARCHAR2(14)
DEFERRED                  VARCHAR2(9)
VALIDATED                 VARCHAR2(13)
GENERATED                 VARCHAR2(14)
BAD                       VARCHAR2(3)
RELY                      VARCHAR2(4)
LAST_CHANGE               DATE
INDEX_OWNER               VARCHAR2(128)
INDEX_NAME                VARCHAR2(128)
INVALID                   VARCHAR2(7)
VIEW_RELATED              VARCHAR2(14)
ORIGIN_CON_ID             NUMBER
```

Search Data Dictionary

STUDENT

```
# studnr
* first_name
* last_name
o cell
```

```
select constraint_name,constraint_type
from user_constraints
where table_name = 'STUDENT'; -- NOTE! CAPITAL LETTERS
```

```
CONSTRAINT_NAME                CONSTRAINT_TYPE
------------------------------ ---------------
SYS_C0010189                   P
SYS_C0010187                   C
SYS_C0010188                   C
```

```
CONSTRAINT_TYPE (from 12c docs)
C - Check constraint on a table
P - Primary key
U - Unique key
R - Referential integrity
V - With check option, on a view
O - With read only, on a view
```

The name of our PK is: SYS_C0010189

Naming constraints

STUDENT

```
# studnr
* first_name
* last_name
o cell
```

STUDENT

studnr	first_name	last_name	cell
1	erik	andersson	0733359556

SYS_C0010189 is not a good name on a constraint.

```
insert into student(studnr,first_name,last_name,cell)
values('1','carina','eriksson','0733358559');
```

Error starting at line : 1 in command -
insert into student(studnr,first_name,last_name,cell)
values('1','carina','eriksson','0733358559')
Error report -
SQL Error: ORA-00001: Unique constraint violated
(TEST.SYS_C0010189) This name tells us nothing!
00001. 00000 - "unique constraint (%s.%s) violated"

Change name on constraints

STUDENT

```
# studnr
* first_name
* last_name
o cell
```

```
alter table student
rename constraint SYS_C0010189 to student_studnr_pk;
```

table STUDENT altered.

It is always good to give your constraints good names. There are different naming conventions. Here is one that I use:

table_column_type

Type	Abbrivation
Primary Key	pk
Foreign Key	fk
Check	ck
Unique	uq

Naming constraints

STUDENT

```
# studnr
* first_name
* last_name
o cell
```

STUDENT

studnr	first_name	last_name	cell
1	erik	andersson	0733359556

We test again to se the difference:

```
insert into student(studnr,fnamn,enamn,mobil)
values('1','carina','eriksson','0733358559');
```

Error starting at line : 1 in command -
insert into student(studnr,first_name,last_name,cell)
values('1','carina','eriksson','0733358559')
Error report -
SQL Error: ORA-00001: Unique constraint violated
(TEST.STUDENT_STUDNR_PK)
00001. 00000 - "unique constraint (%s.%s) violated"

Drop a constraint

STUDENT

```
# studnr
* first_name
* last_name
o cell
```

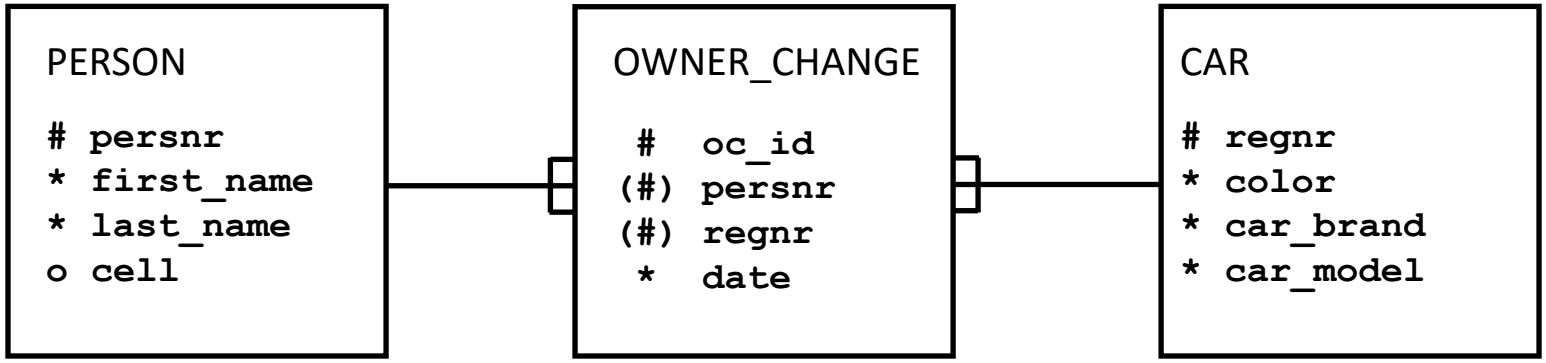
If we want to remove a constraint we can use the drop-statement. We only need to know the name of the constraint:

```
alter table student
drop constraint student_studnr_pk;
```

Now PK is removed. If we want, we can add more constraints at **the same time**. We add our PK again and a unique constraint on the cell column:

```
alter table student
add constraint student_studnr_pk primary key(studnr)
add constraint student_cell_uq unique(cell);
```

Add Foreign Keys



```
create table owner_change(
oc_id number(9),
persnr varchar2(13),
regnr varchar2(6),
date date default sysdate not null);

alter table owner_change
add constraint ownerchange_ocid_pk primary key(oc_id)
add constraint ownerchange_persnr_fk foreign key(persnr) references person(persnr)
add constraint ownerchange_regnr_fk foreign key(regnr) references car(regnr);
```

Add a column to a table

STUDENT

```
# studnr
* persnr
* first_name
* last_name
o cell
```

```
alter table student
add persnr number(10);
```

Oops, there was wrong data type on the column. We want **varchar2**(13) instead so we can store social security numbers as a string on the form 'YYYYMMDD-NNNN'.

We can change the data type of the column with the following SQL statement:

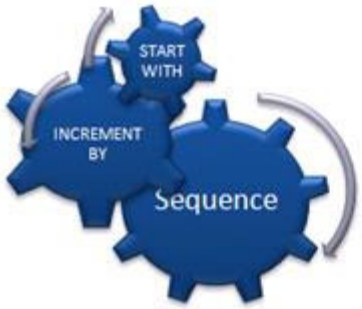
```
alter table student
modify persnr varchar2(13);
```

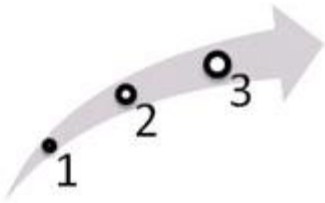
The Sequence object

OWNER_CHANGE

```
 #   oc_id
(#)  persnr
(#)  regnr
 *   date
```

112568, 112569...



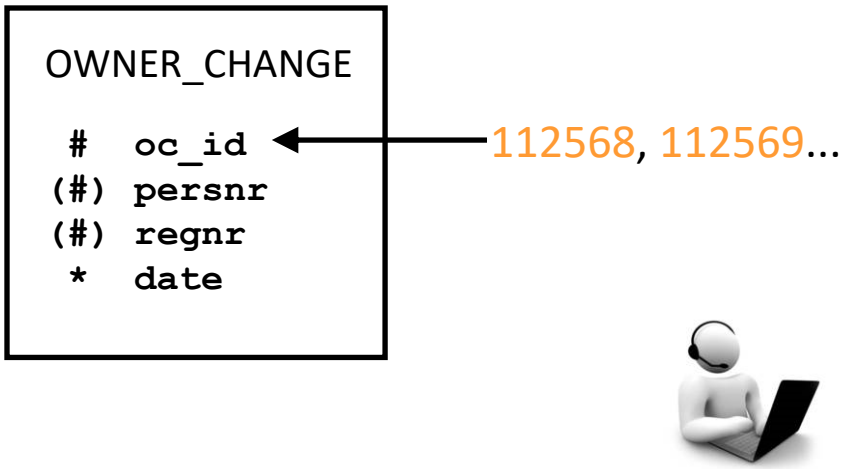


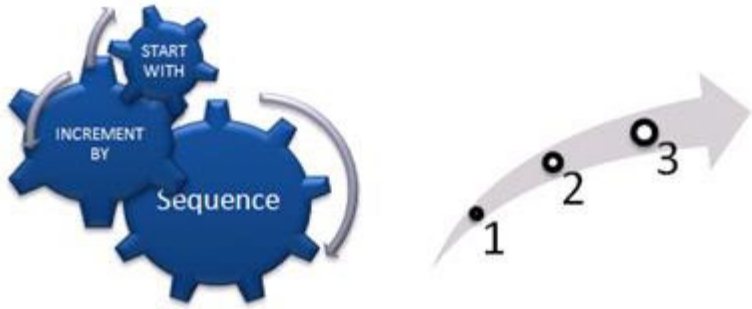
It may be a problem to generate new unique values for the PK in OWNER_CHANGE table if we leave this to the administrators at the Swedish Road Administration.

Better to let a sequence fix this!

A sequence is a database object that we can see as a black box. Each time we call the *nextval* function on the sequence object we will get a new unique number.

The Sequence object





We can create a sequence like this:

```
create sequence seq_ownwerchange
start with 1
increment by 1;
```

Test if it works:

```
select seq_ownerchange.nextval
from dual;


 NEXTVAL
----------
         1
```

How do we use the sequence?

OWNER_CHANGE

oc_id	persnr	regnr	date
1	19940610-7140	ABC567	2019-11-05

We call the sequence during the insert. We use SQL DML:

```
insert into owner_change(oc_id,persnr,regnr,date)
values(seq_ownerchange.nextval,'19940610-7145','ABC567',sysdate);
```

The Dual table

DUAL is a table automatically created by Oracle Database along with the data dictionary. DUAL is in the schema of the user SYS but is accessible by the name DUAL to all users. It has one column, DUMMY, defined to be VARCHAR2(1), and contains one row with a value X.

Selecting from the DUAL table is useful for computing a constant expression with the SELECT statement. Because DUAL has **only one row**, the constant is returned only once.

DUAL

* dummy

```
select * from dual;

DUMMY
-----
    X
```

Examples

```
select 'Kalle Andersson'
from dual;
```

```
'KALLEANDERSSON'
----------------
Kalle Andersson
```

```
select sysdate
from dual;
```

```
SYSDATE
----------
2019-11-10
```

```
select sysdate + 10
from dual;
```

```
SYSDATE + 10
----------
2019-11-20
```

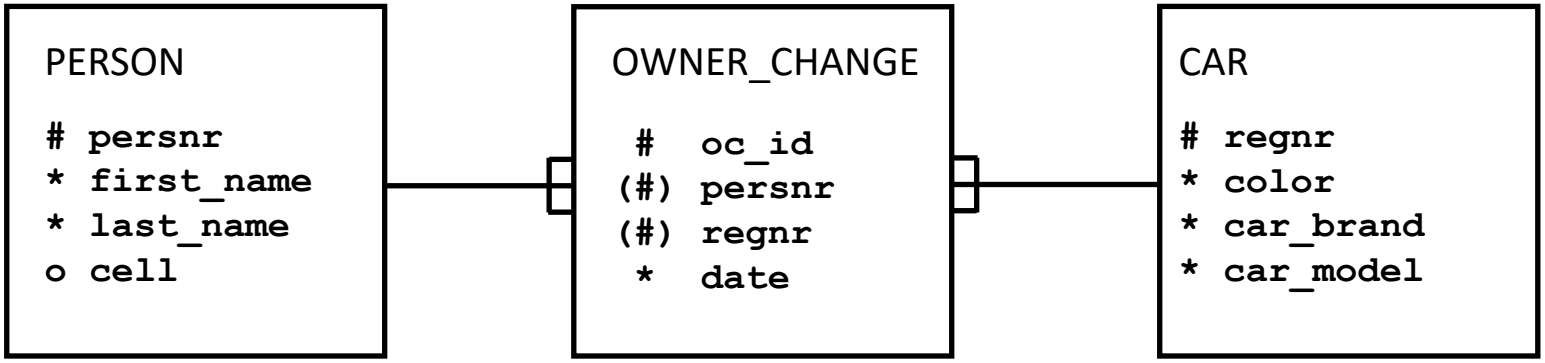
```
select 1 + 2 + 3
from dual;
```

```
1+2+3
-----
6
```

Remove an object

To remove an object we can use:

```
drop object_name;
```



If we are to remove the above tables, then this must be done in a *certain order*:

1. `drop table owner_change;`
2. `drop table person; drop table car;`

We cannot drop person or car first. As this causes **referential integrity violation**.
We would have FK values in owner_change that **lacked references**.

Data types - simple data types

"I have a very simple rule: **Put dates in dates**, **numbers in numbers,** and **strings in strings**. Never use a datatype to store something other than what it was designed for and use the most specific type possible. " (Tom Kyte, Oracle)

Varchar2 when we want to store text. Remember to put single quotation marks ' around the text in 'a text string'. We can set the maximum length by adding the string length, **varchar2**(25) has a maximum length of 25 characters.

Number when we need to store numbers: 345, 3, 67.3, 3.14. We don't put single quotation marks around numbers. number(3) means a maximum of 3 digits, e.g. 999. number(5.2) means total length of **5** digits including 2 decimal places, e.g. 145.25

Date when we need to store times. Distinguish how it is stored in the database and how it is presented. If we store sysdate, then we have year, day, hour, minute and second. We can present it as'2024-01-25'or '01 -JAN-2024' by specifying a certain format mask. We put single quotation marks' around dates '2016-12-24'.

Conversion Functions

Various **built-in functions** that handle the **conversion of data types**.

to_char, **to_date**, **to_number** **etc.**

We can convert a date, e.g. sysdate to a **string** in the desired format 2027-10-01:14:15:58.

We test with a SQL statement :

```
select to_char(sysdate,'YYYY-MM-DD:HH24:MI:SS') as right_now
from dual;
```

```
RIGHT_NOW
-------------------
2019-11-05:18:52:43
```

Do not compare apples with pears

Sometimes we may need to convert a 'character string' to a date or a date to a 'character string'.

We may want to check if there are any rows in the customer table with a reg date equal to '2019-05-25'.

```
where regdate = '2019-05-25';
```

date string

Furthermore, only compare **dates to dates**, **strings to strings**, and **numbers to numbers**. When dates and numbers are stored in strings, or stored using inappropriate lengths, **your system suffers:**

1. When you insert data into the database, you lose the ability to check that dates are real dates, that numbers are real numbers**.**
2. The database loses performance.
3. Data integrity and data quality are declining.
4. Your system suffer!

Compare apples with apples

We must ensure that the **same data type on both sides** of the equal sign **=**

```
where to_char(regdate,'YYYY-MM-DD') = '2019-05-25';
```

string string

If we want to check how many who registered in 2019 :

```
where to_char(regdate,'YYYY') = '2019';
```

If we want to check how many who registered in August 2019 :

```
where to_char(regdate,'YYYY-MON') = '2019-AUG';
```

Consequences of incorrect data types

If we store a **date** as a *number* or a *string*, this can have serious consequences!

What prevents me from entering **2020-02-96**?

The 96th of February does not exist. Well in our database it does. This is an example of *low data integrity*.

What prevents me from entering **2017-E2-A6**?

2017-E2-A6, what day is that?

If we had set **dates as dates**, this would not have been possible. The system had stopped us from entering our junk data into the database.

"Put dates in dates, **numbers in numbers,** and **strings in strings**."

Operations on the data type date

date + number = date

date - number = date

date - date = number the number of days between two dates

```
select to_date('JAN-01-2026','MON-DD-YYYY') + 10 as return_date
from dual;
```

```
RETURN_DATE
----------------
2026-01-11
```

```
select sysdate - to_date('1993-02-23','YYYY-MM-DD')
from dual;
```

```
sysdate - to_date('1993-02-23','YYYY-MM-DD')
-------------------------------------------
8052,74644
```

Round decimals

We can use the function **round()** to round to the desired number of decimals:

```
select round(sysdate - to_date('1993-02-23','YYYY-MM-DD'),0) as numberOfDays
from dual;
```

```
NUMBEROFDAYS
------------
8053
```

More about **Oracle built in functions**:

http://www.techonthenet.com/oracle/functions/