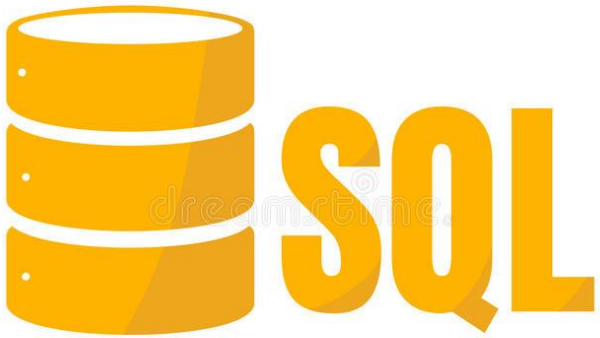
Introduction to SQL



General scheme of presentation

- SQL
  - Introduction, history, concepts
  - DML
  - DDL
  - DCL
  - Advanced topics:

SQL - Introduction

- Structural Query Language is a standardized language for Databases. Basically, It's a declarative language (not procedural).
- SQL is born in 1974 (called SEQUEL). SQL has been standardized in different versions (see next slide) by ANSI and ISO.
- The implementations of SQL for the different RDBMS (Oracle, Sybase, SQL Server, Ms-Access, mySQL) are not always standard.
- As many languages, SQL is defined by a dictionary and a grammar.
- In this training session, we will use the SQL92 syntax (SQL2).

SQL history

- Extract of http://en.wikipedia.org/wiki/SQL

Year	Name	Alias	Comments
1986	SQL-86	SQL-87	First published by ANSI. Ratified by ISO in 1987.
1989	SQL-89	FIPS 127-1	Minor revision, adopted as FIPS 127-1.
1992	SQL-92	SQL2, FIPS 127-2	Major revision (ISO 9075), *Entry Level* SQL-92 adopted as FIPS 127-2.
1999	SQL:1999	SQL3	Added regular expression matching, recursive queries, triggers, support for procedural and control-of-flow statements, non-scalar types, and some object-oriented features.
2003	SQL:2003		Introduced XML-related features, *window functions*, standardized sequences, and columns with auto-generated values (including identity-columns).
2006	SQL:2006		ISO/IEC 9075-14:2006 defines ways in which SQL can be used in conjunction with XML. It defines ways of importing and storing XML data in an SQL database, manipulating it within the database and publishing both XML and conventional SQL-data in XML form. In addition, it provides facilities that permit applications to integrate into their SQL code the use of XQuery, the XML Query Language published by the World Wide Web Consortium (W3C), to concurrently access ordinary SQL-data and XML documents.

SQL

- SQL is composed of different sub-languages:
  - DDL : data definition language
  - DML : data manipulation language and transaction control
  - DCL : data control language, definition of rights

- To learn those languages, we first have to cover some items:
  - naming
  - data types
  - literal values (constants)
  - functions

SQL Naming

- You will mainly define names for
  - Table & views
  - Table fields & aliases
- There are some naming limits (it may differ from a DB to another):
  - Usable chars : [A-Z] for the first char, [A-Z0-9_] for next chars
  - Length of names : 30 chars for Oracle and DB2 fields names
  - Character case in not significant.
  - You can’t use reserved names (tokens of the SQL language).
- You should respect supplementary constraints in naming:
  - Public conventions: for example, do not prefix names of fields with the name of the table.
  - Specific conventions.

SQL – Data types

- Main data types:
- CHAR(len): fixed-length string
- VARCHAR2: variable-lengh string
- NUMBER(precision,scale): decimal or float number. Precision = total number of digits. Scale = number of digits after the decimal.
- DATE : calendar date (in Oracle, this type will include the time).
- TIMESTAMP: Date and time
- INTERVAL : interval of time
- BLOB: Binary large object

SQL – constants (literals)

- Numeric constants: written as usually; for example : 123 or -1245.63
- String constants: enclosed in simple quotes; for example ‘hello world’. If you have to include a simple quote in your string, you have to write two consecutive simple quotes; for example : ‘I can’’t do it!’
- Date constants: DATE ‘yyyy-mm-dd’; example : DATE ‘2001-12-13’. There is no syntax for a date literal with a time, you may use the “TO-DATE” function to convert a string; for example: TO_DATE('98-DEC-25 17:30','YY-MON-DD HH24:MI')
- Timestamp constants: TIMESTAMP ‘yyyy-mm-dd HH:MM:SS’
  - ; example TIMESTAMP ‘2007-01-01 12:34:56'
- Interval literal: INTERVAL ‘n[-n]’ [(YEAR|MONTH)] [‘(‘n’)’] [TO MONTH]. Example: INTERVAL '123-2' YEAR(3) TO MONTH

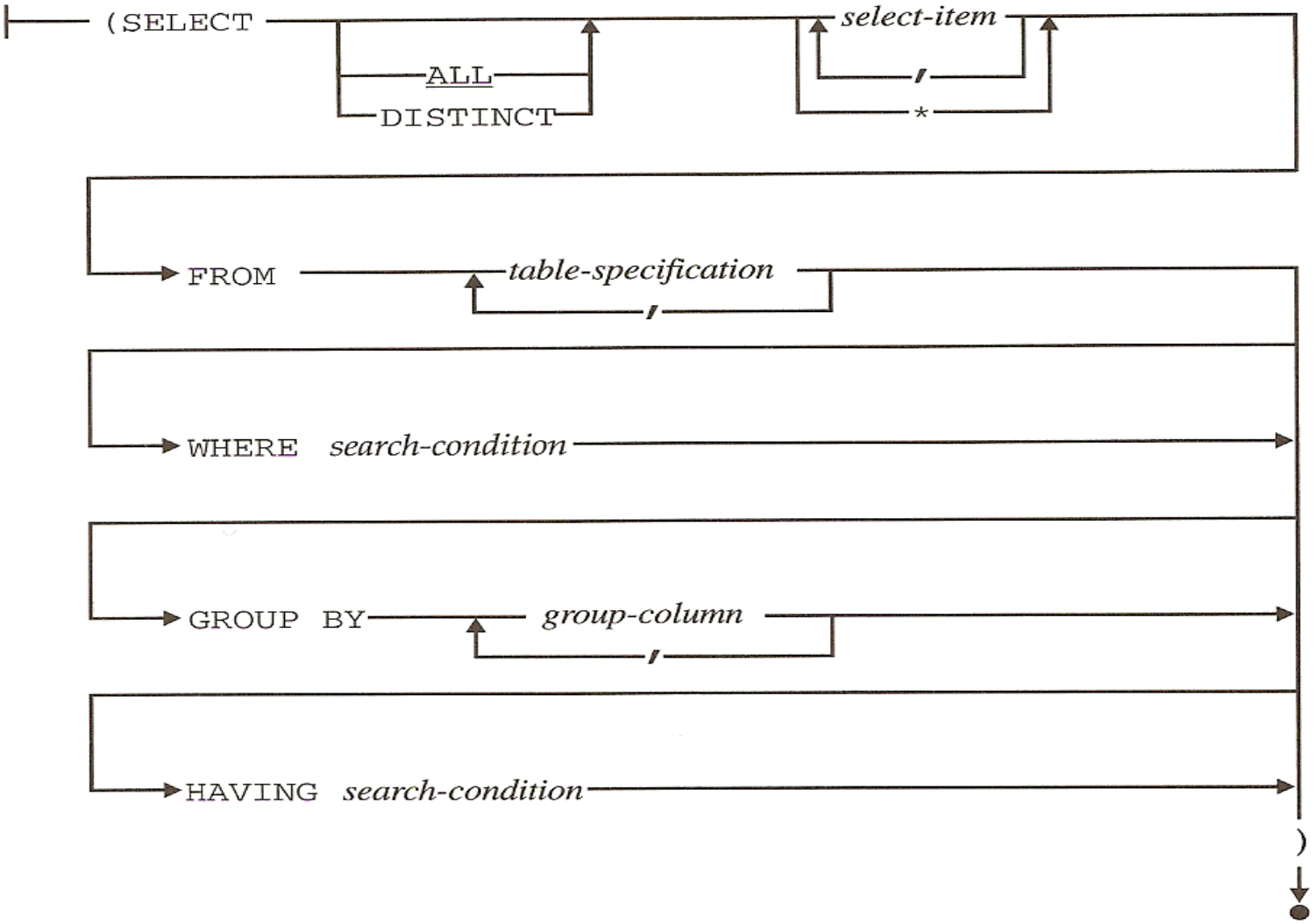
SQL built-in functions

- SQL defines standard functions. Each RDBMS may offer additional functions.
- We may distinguish different types of functions:
  - Get context info: current_time, current_date, current_timestamp, current_user, session_user, system_user. (not covered by Oracle)
  - Aggregate functions: max, min, count, avg, sum
  - Functions to manipulate strings: char_length, *position(starting_string IN search_string),, lower, upper, concatenate, trim, …*
  - Date & time functions: year, month, day, hour, minute, second, …
  - Mathematics functions: sin, cos, ceil, floor, log, ln,…
  - Conversion of types: cat (expression as type)

SQL - DML

- Select : select rows of one table or of two or more related tables.
- Insert : insert a new row in a table.
- Delete : delete one or more rows in a table.
- Update : update one or more rows in a table.

SQL DML – Query syntax



SQL DML - Query

- Structure of our presentation on the queries:
  - Simple queries
  - Ordering results (order by)
  - Sub-queries
  - Distinct results
  - Aggregations
  - Multi-tables queries

SQL DML - Query

- Main elements of the syntax of a simple query:

select *selectList* from *tableName* where *clause*

- Some examples:

select * from cld01;

select NOMCLI, TELPRI from cld01;

select NOMCLI, TELPRI from cld01 where TELPRI not like ' %';

SQL SIMPLE QUERY – select list

- selectList = “*” or “fieldName *(,* fieldName*)**” or function(fieldName) or calculation expression or a literal.
- “fieldName” is here the name of one of the fields defined in the target table or the name of a pseudo field.
- You may defined aliases for column names, syntax: fieldListElement as aliasName
- Examples:

```
select NOMCLI, TELPRI, ROWNUM from cld01

select '++', NOMCLI, SUBSTR(NOMCLI,1,1) from cld01

select NOMCLI,  SUBSTR(NOMCLI,1,1) as ini from cld01
```

- Trick: to see the fields defined in a table, use “desc tableName”.

SQL SIMPLE QUERY – select list - operators

- In SQL select list and in SQL clauses, you may use some operators:
  - mathematical operators: +, -, *, /, %
  - string concatenation: ||

- Examples:

```
select DATMAJ-DATCRT as daysBeforeCreationAndLastUpdt,
DATCRT, DATMAJ, NOMCLI from cld01

select nomcli || ‘ ‘ || prncli as nomprncli from cld01
```

SQL SIMPLE QUERY – where clause

- Where clause = WHERE *condition ((*AND|OR*)* *condition)?*
- Simple Condition = *(*NOT*)? fieldName operator (* *fieldName* | *litteralValue)?*
- Operators:
  - Comparison: =, <>, <, >, <=, >=
  - Range test: testExpr *(*NOT*)?* BETWEEN lowValExpr AND highValExpr
  - Pattern matching condition: LIKE pattern
    - wildcards chars: '%', _ - You may use an escape char but you have to define it.

SQL SIMPLE QUERY – where clause ...

- Examples:

select NOMCLI from cld01 where NOMCLI LIKE 'D_s%‘

select NOMCLI from cld01 where NOMCLI LIKE '%_%' ESCAPE '\'

select DATCRT, NOMCLI from cld01 where DATMAJ-DATCRT =0

SQL SIMPLE QUERY – where clause …

- To test a “null” value; use “is null” expression or “is not null” or “not fieldName is null”.
- Example:

select NOMCLI from cld01 where NOMCLI is null

SQL SIMPLE QUERY – where clause …

- To group conditions, you will use “and” or “or” operators and parenthesis to modify default priority in the order of evaluation of operators.
- Example:

```
select  mnecli, nomcli, nomalp from CLD01
where (mnecli like 'R%' or nomcli like 'R%')
and  length(NOMCLI)=9;
```

SQL SIMPLE QUERY – ordering results

- Syntax: select … order by *expression (*, *expression (*ASC|DESC*)?)**
- Examples:

```
select  mnecli, nomcli, nomalp
from CLD01
order by nomalp;

select  nomalp
from CLD01
order by DATCRT DESC, DATMAJ;
```

SQL Query & sub-queries

- SELECT … WHERE *fieldName* IN (SELECT *fieldName* from …)
- SELECT … WHERE *fieldName* NOT IN (SELECT *fieldName* from …)
- SELECT … WHERE *(*NOT*)?* EXISTS (SELECT …)
- SELECT … WHERE *fieldName comparisonOp* (SELECT *fieldName* from …)
- SELECT ... WHERE *fieldName comparisonOp (*ANY|ALL*)* (SELECT *fieldName* from ...)
- Examples:

```
select  nomalp from CLD01
where length(nomalp) = (select max(length(nomalp)) from CLD01)

select count(*) from CLD01
where numcli not in( select numcli from CLD02)

select count(*) from CLD01 x
where exists( select * from CLD02 where numcli=x.numcli)
```

SQL Query - Distinct results

- Syntax: select distinct …
- Examples:

```
select distinct devref
from CLD01

select distinct codbic, numprc
from CLD01
```

SQL Query - Aggregations

- General Syntax: select … from … group by …. *(having condition)*
- In the field list and in the “having” condition, you have to use aggregation functions except for the fields on which the group is formed.
- You may group on several columns.
- You may add a condition to filter groups with the “having condition”.

SQL Query - Aggregations

- Examples:

```
select devref, count(*) from CLD01 group by devref

select devref, count(*) as c from CLD01
group by devref order by c

select devref, max(monact) as M, sum(monact) as S, (max(monact)/
sum(monact)*100) as R from CLD01
group by devref having sum(monact)<>0

select devref, nbremp, max(monact) as M, count(*) as c from CLD01
group by devref,nbremp
having (sum(monact)<>0 )and (nbremp>1)

select devref, max(nbremp), max(monact) as M, count(*) as c
from CLD01 where nbremp > 1
group by devref having (sum(monact)<>0 )
```

SQL Query – Aggregate functions defined in SQL

- SQL defines some aggregate functions:
  - AVG : average of values, only for numeric types
  - COUNT : number of values
  - MAX : maximum value
  - MIN : minimum value
  - SUM : sum of values, only for numeric types
- Null values are ignored in aggregate functions. In oracle, you mask ask to consider null values with a NVL function (by example `avg(nvl(sold,0))`).
- With the count function, you may use the “count(*)” syntax to count the number of rows.
- If you want to eliminate duplicates values before the execution of the aggregate function, you will use the syntax “fct(distinct fieldName)”.

SQL Query – usage of aggregate functions

- You may use aggregate functions in SQL queries having no “group by” clause. You have then a group composed of all records returned by your select.
- Example:

```
select count(*), sum(SOLPCD)
from CCD01
```

SQL Multi-tables queries (joins & unions)

- Joins : rows of the result set have fields coming from two (or more) tables.
  - cartesian product (or cross join): it produces n * m rows (with n= number of rows of table 1, m = number of rows of table 2).
  - equi-join: returns only rows combining rows of the two tables having same values in fields participating to the relation.
  - left outer-join: all records of the left table are present in the result set even if the right table has no corresponding record (in this case, the fields coming from the right table have null values).
  - right outer join: all records of the right table are present in the result set even if the left table has no corresponding record (in this case, the fields coming from the left table have null values).
- Unions: rows returned by two select with the same structure are added.

SQL joins

- Select … from tbl1, tbl2

⇒We obtain n * m rows (where n=number of rows of tbl1 and m=number of rows of tbl2) with the sum of fields of tbl1 and tbl2.

Select … from tbl1, tbl2 where tbl1.fieldX=tbl2.fieldY

⇒We obtain related rows of tbl1 and of tbl2 with the sum of fields of tbl1 and tbl2.

Example :

```
select * from CLD01,CCD01
where CCD01.numcli = CLD01.numcli
```

SQL outer join

- As in inner join, we join 2 tables and we select all records of the first table and corresponding records of the second table. But here, if no corresponding records exists in the second table we nevertheless return a result record where fields from the second table receive the null value.
- Example:

```
select * from CCD01 left outer join CCD51
where CCD01.natenc=CCD51.natenc
```

SQL select – combination of sets: Union, intersection, difference

- We may ask the SQL engine to combine rows of different queries:
- Union: combination of two or more queries; we add rows of different queries giving compatible fields.
- Intersect : combination of two or more queries; we keep common rows of different queries giving compatible fields.
- Except (standard SQL), minus (Oracle); we keep only rows returned by the first query and not by the second.

SQL select – combination of sets: Union

- Union: combination of two or more queries; we add rows of different queries giving compatible fields.
- Syntax:

SELECT … UNION *(*ALL*)?* SELECT …

- Example:

```
select 'NoAdress' ST, numcli
from CLD01 X
where numcli not in( select numcli from CLD02)
union
select 'withAdresses' ST, numcli
from CLD01 X
where numcli in( select numcli from CLD02) order by 1
```

SQL select – combination of sets: intersection

- Intersect : combination of two or more queries; we keep common rows of different queries giving compatible fields.
- Syntax: SELECT … INTERSECT SELECT …
- Example:

```
select numcli
from CLD01
intersect
select numcli
from CLD02
```

SQL select – combination of sets: difference

- Except (standard SQL), minus (Oracle): we keep only rows returned by the first query and not by the second.

- Syntax: SELECT … EXCEPT/MINUS SELECT …
- Example:

```
select numcli
from CLD01
minus
select numcli
from CLD02
```

SQL DML - INSERT

- Two different syntaxes:
  - SQL INSERT INTO *tableName* (*fieldName (,* fieldName*)**) VALUES (*literal (, literal)**)
  - SQL INSERT INTO tableName (*fieldName (,* fieldName*)**) select ...
- Examples:

```
insert into AWE_COMPREV (company,year,quarter,revenue)
values('BMW', 2001, 'Q1',125);

insert into AWE_COMPREV values('BMW', 2001, 'Q1',125);

insert into AWE_COMPREV2 select * from AWE_COMPREV;
```

SQL DML - DELETE

- Syntax:
  - DELETE FROM tableName *(* WHERE ... *)?*
- Example:

```
delete from AWE_COMPREV2
where company='IBM';
```

SQL DML - UPDATE

- Update the content of one or more fields of all/selected records.
- Syntax:

UPDATE *tableName* SET *fieldName = expression (* , *fieldName = expression)* (* WHERE ... *)?*

- Examples:

```
update AWE_COMPREV2
set revenue = revenue * 1.25;

update AWE_COMPREV2
set revenue = revenue * 1.25, COMPANY='BMW-BE'
where company='BMW';
```

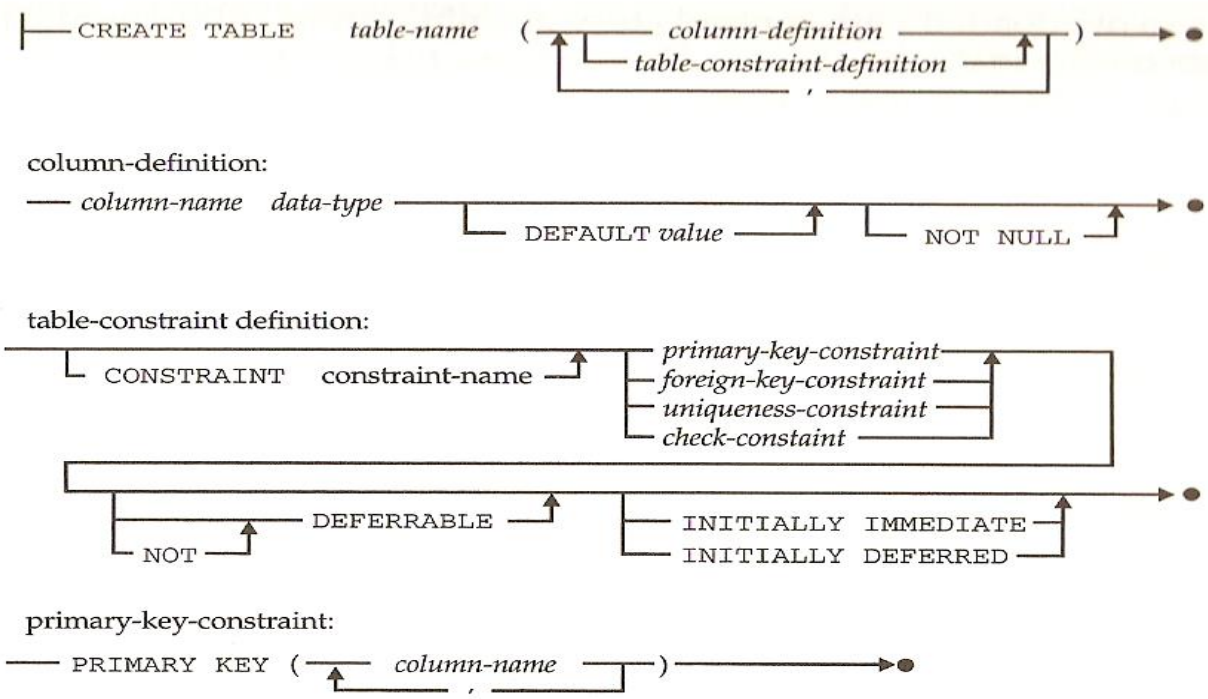
SQL - DDL

- With DDL, Data Definition Language, we can create
- Schemas
- Tables with constraints
- Constraints
- Domains for values (not supported by Oracle, we have to use user defined types instead).

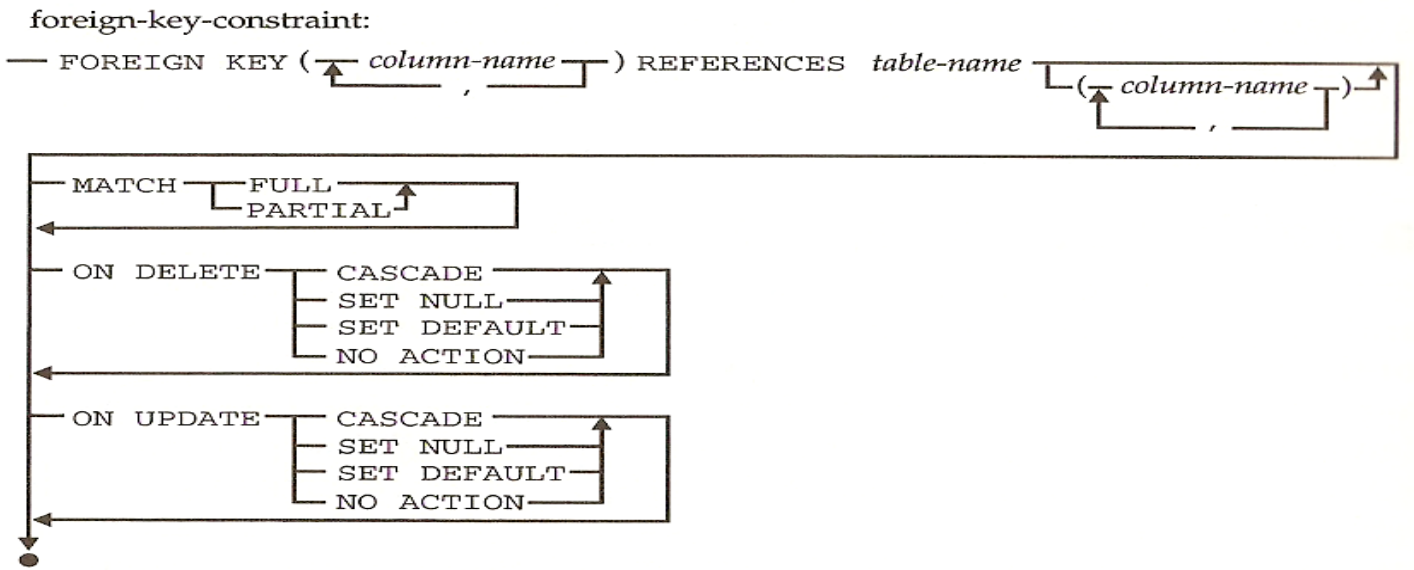
SQL DDL

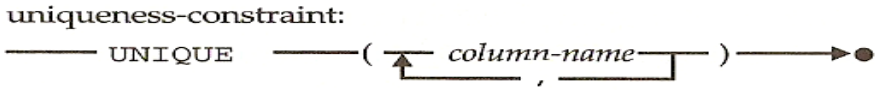
- Create/drop/alter table
- Create/drop view
- Create/drop index
- Create/drop schema
- Create/drop/alter domain

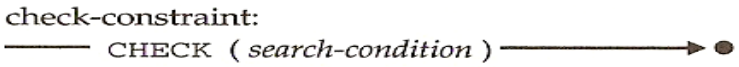
SQL DDL – Create table syntax



SQL DDL Create table syntax ...







SQL DLL - Create table - examples

```
create table employee(name varchar2(100), age int,
salary number, primary key (name)) ;

create table employeeAddr(emp_name varchar2(100),
country varchar2(30), city varchar2(30), street varchar2(200),
house_number int, foreign key (emp_name) references
employee)

create table cld01_1973 as select * from cld01 where datnai
like '1973%';
```

SQL DDL – Create view

- Syntax:

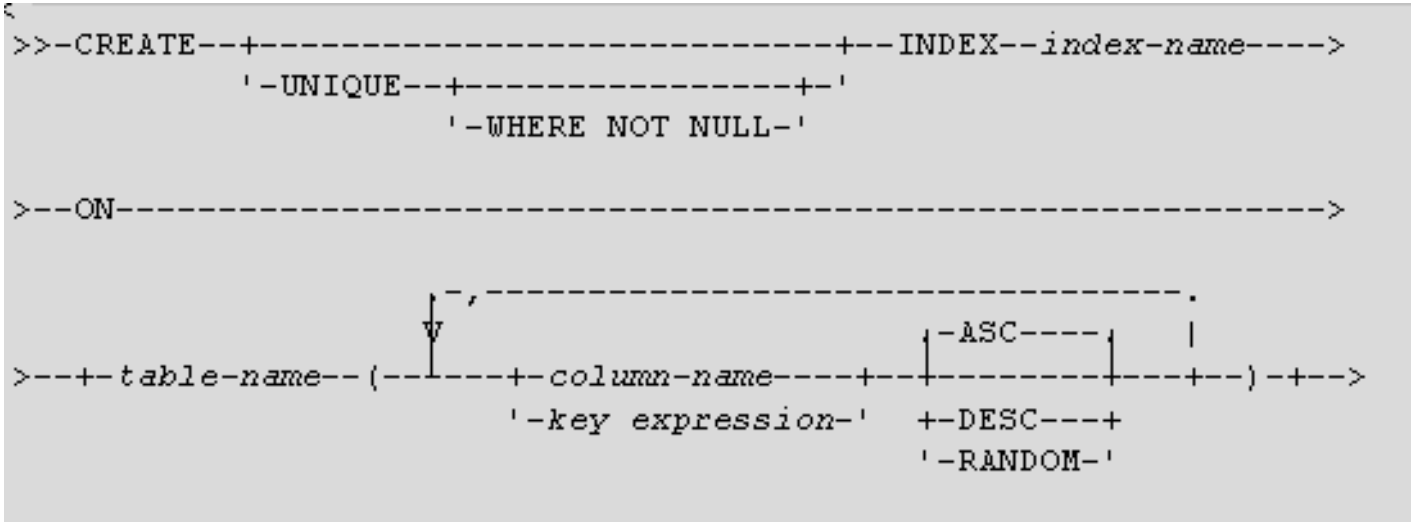
CREATE VIEW *viewName* (
columnName1,
columnName2,
…
) AS
sqlSelectStatement

- *Example:*

```
create view AWE_COMP_YEAR_REV as
select company,year,sum(revenue) as yearRev
from AWE_COMPREV
group by company,year;
```

Create index

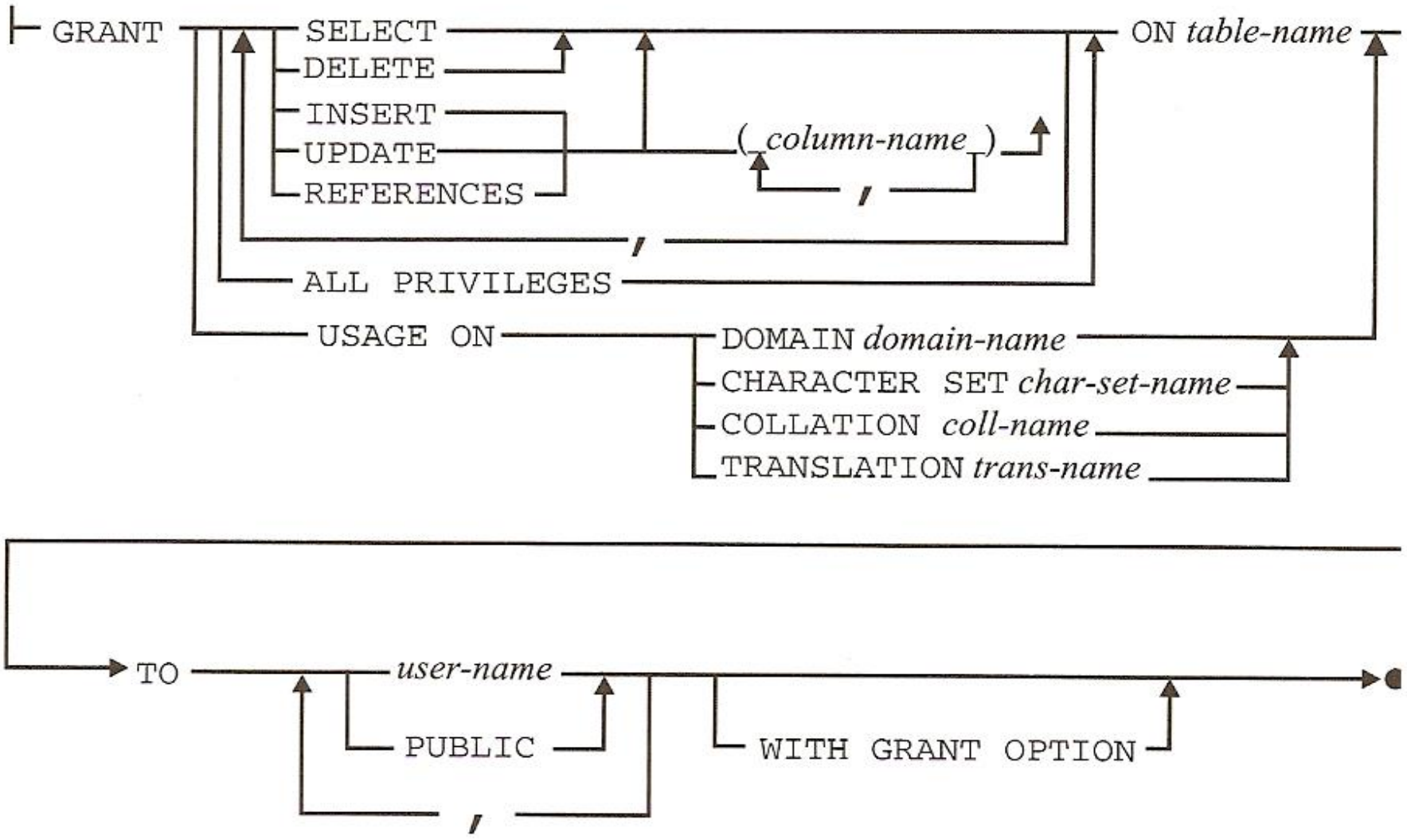
- The create/drop index statements are not part of the standard SQL but are used by Oracle and DB2.
- Ask the creation of an index on one or more fields of a table. You may specify an uniqueness constraint.
- Basic syntax:



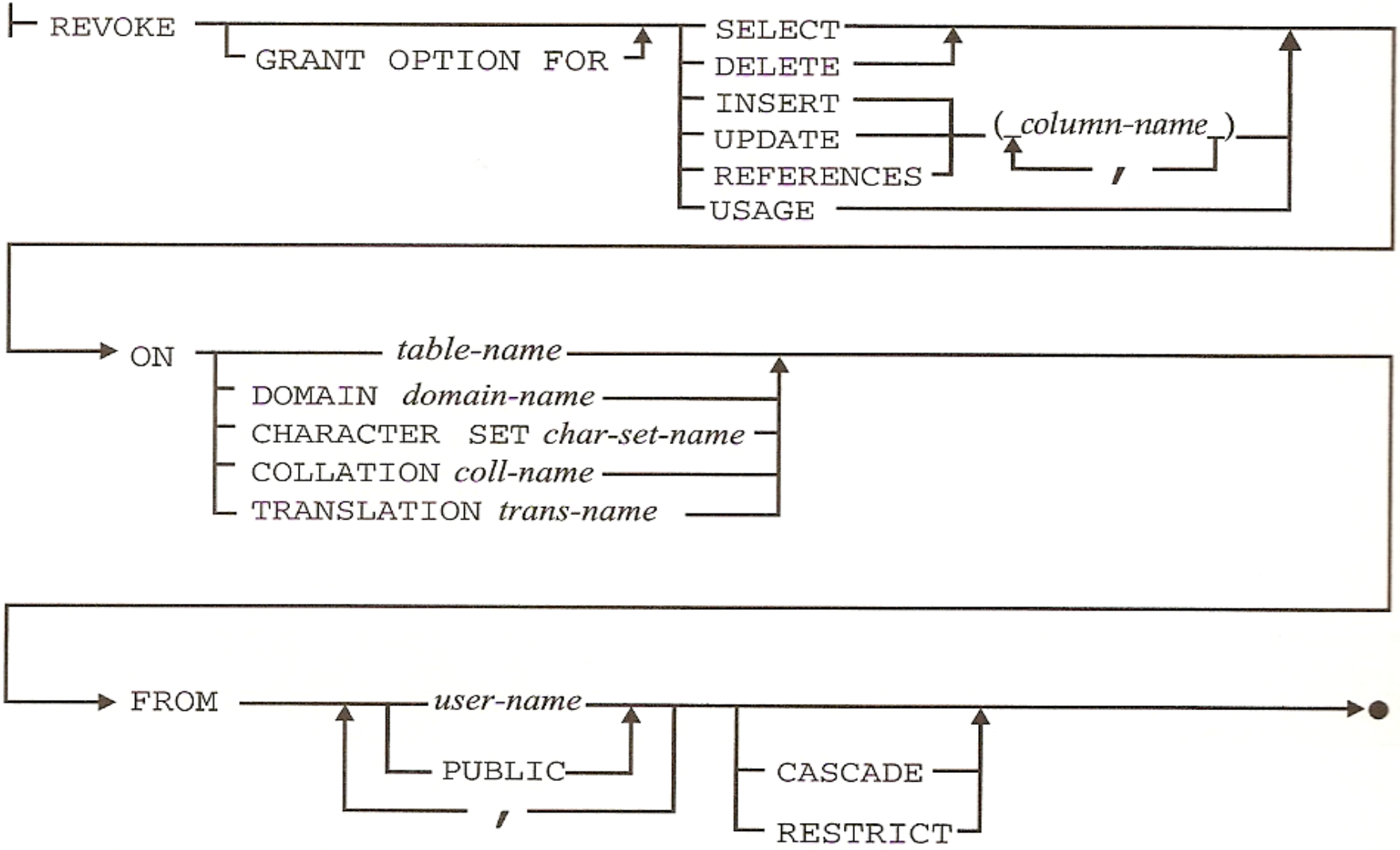
SQL DCL – Access control

- The SQL norm defines some security concepts:
- Each user receives privileges on database objects.
- Privileges are allowed actions : select, delete, insert, update, reference (the right to add a reference to a row) , grant (the right to pass your rights)
- Privileges are given or removed using the SQL grant/revoke instructions.
- Privileges are given to individual users or to everybody (PUBLIC).
- When you create a new table, you are the owner of this table and not any other user has access to it. Of course, you may give privileges to other users.

SQL DCL - Grant



SQL DCL - Revoke



Transactions

- Different users use concurrently different sessions. A single user can have more than one session.
- A transaction is a sequence of SQL statements that the RDBMS treats as a single unit of work
- In each session, the user can use successive transactions (you can use only one transaction at the same time in the same session). A transaction is finished by a “commit” or a “rollback”. You can also use savepoint/rollback to savepoint to isolate parts of the transaction with the possibility to “undo” only effects of statements in those parts. A savepoint is a marker within a transaction that allows for a partial rollback

Transactions

- The session can be configured to work in “auto-commit” mode (On Oracle, with SQL Plus, you modify the mode with the command “SET AUTOCOMMIT (ON|OFF)” .
- Isolation levels & explicit locking will give the behavior of concurrent sessions concerning the not-committed modifications of the data.

Transactions – ACID properties

- A transaction should guarantees some properties:
- Atomicity: each transaction is validated (or invalidated) as a all.
- Consistency: a consistent result is obtained a the end of the transaction.
- Isolation: effects of other concurrent transactions are not visible on each individual transaction.
- Durability : effects of each transaction are stored when the transaction is validated.

SQL – Transaction control

- Commit
- Rollback
- Savepoint <name>
- Rollback to savepoint <name>
- Set transaction

Necessity of isolation levels

- Isolations levels offer different answers to possible problems related to the concurency of different transactions:
  - Dirty read : you read data not already commited by concurrent transactions.
  - Non-repeatable read : in the same transaction, when you do two times the same select, you can obtain different results in already read records due to committed updates from other transaction.
  - Phantom read: in the same transaction, when you do two times the same select with a where clause, you can obtain more records due to the insertion of new records from other transactions.

Isolation levels defined in SQL

- Serializable: the upper isolation level; no dirty read, no non-repeatable read, no phantom read.
- Repeatable read: only phantom reads are possible.
- Read commited: non repeatable reads and phantom reads are possible.
- Read uncommitted: dirty reads, non-repeatable reads, phantom reads are possible.

SQL Locking

- Automatically in function of the isolation level.
- Explicitly through:
  - Select … from … for update
  - LOCK TABLE <table_name> IN <lock_mode> MODE [NOWAIT | WAIT <seconds>];

  The lock remains until you close the current transaction.

Triggers

- A *trigger* defines a set of actions that are performed in response to an insert, update, or delete operation on a specified table. When such an SQL operation is executed, the trigger is said to have been *activated*.
- CREATE TRIGGER <name> <action> ON <table_name> <operation> <triggered_action>
- Example (for Oracle DBMS):

```
CREATE TRIGGER NEW_HIRED
AFTER INSERT ON EMPLOYEE
FOR EACH ROW
UPDATE COMPANY_STATS SET NBEMP = NBEMP+1;
```

System catalog

- The meta data used by the RDBMS to store information about your tables, your views, your rights, … are consultable through different tables or views:

DBMS	Tables	Columns	Users	Views	Privileges
Oracle	user_catalog	user_tab_columns	all_users	user_views	user_tab_privs
	user_tables	all_tab_columns		all_views	User_col_privs
	all_tables				User_sys_privs
	user_synonyms				
DB2	Schemata	columns	dbauth	Views	Schemaauth
	Tables			viewdep	Tabauth
	References				Colauth
	keycoluse				

Programming with SQL

- Using direct SQL statements through a tool (for example va SQL PLUS).
- Using embedded SQL in an host language (for example SQL orders embedded in C code).
- Using SQL with an API (for example, via ODBC or JDBC).
- Using SQL instructions in stored procedures (for example, in PL/SQL for Oracle) – such procedures can be launched by triggers or by API calls.
- Using OR Mapping Tools

Stored procedures

- Stored procedures are written in a programming language and executed by the RDBMS .
- Oracle can load and execute stored procedures written in PL/SQL or in Java.
- DB2 has his own language similar to PL/SQL for stored procedures
- The main advantage of using stored procedures is to limit the network traffic and by this way to obtain good performances.

Stored procedures

- A second advantage is the possibility to create a separated layer (separated from the client application) hiding the knowledge of the data structure.
- Stored procedures can be executed by triggers, by SQL queries (via PL/SQL functions), by API calls.

Thanks for your attention!