Centre de Calcul de l’Institut National de Physique Nucléaire et de Physique des Particules

SQL Databases Training

January 22 -25, 2019

Osman Aidel – Philippe Cheynet

Course Content

- DB history (30 min)
- Relational Model (3h)
- SQL (3h)
- PL/SQL (4h)
- Advanced Objects (3h)
- Optimization (7h)

Course Content

- DB history (30 min)
- Relational Model (3h)
- SQL (3h)
  - Basic Information
  - Advanced Information
- Advanced Objects (3h)
- PL/SQL (4h)
- Optimization (7h)

SQL – Advance – Content

- SQL History
- Types
- Languages
  - DML
  - DDL
  - DCL
  - TCL
- Advanced Usage

SQL – Advance – Content

- DCL (user privileges)
- TCL (Transaction Control)
- Advanced Types
- The NULL value
- The WITH Clause
- Joining Data
- Grouping Sets
- SubQueries

And About Indexes ?
And Yes, No reference to index in SQL Standard
It’s a technical object use by RDBMS to
Improve performance.
Details in the next section (advanced objects)

SQL – Alias

An alias assigns a table or a column a temporary name in a query.
The aliases only exist during the execution of the query.

SELECT column_name/expression AS alias_name FROM table [AS] alias_name

```
SELECT
    first_name || ' ' || last_name
FROM
    customer
ORDER BY
    first_name || ' ' || last_name;
```

```
SELECT
    first_name || ' ' || last_name AS full_name
FROM
    customer AS c
ORDER BY
    full_name;
```

SQL – Part 1 – Commands

DCL Commands **D**ata **C**ontrol **L**anguage

DCL commands are used to enforce database security in a multiple user database environment.

Only Database Administrator's or owner's of the database object can provide/remove privileges on a database object.

Command	Description
GRANT	Gives a privilege to user on a database object
REVOKE	Takes back privileges granted from user

Roles:

easier way to GRANT or REVOKE privileges to the users through a role rather than assigning a privilege directly to every user.

SQL – Command

GRANT

SQL GRANT is a command used to provide access or privileges on the database objects to the users.

```
GRANT
   privilege_name
ON
   object_name
TO
   {user_name |PUBLIC |role_name}
[WITH GRANT OPTION]
```

select, insert, update, delete, truncate
temporary, trigger
create, connect, usage, execute
ALL [privileges]

table, column, view, foreign table, sequence, database, foreign-data wrapper, foreign server, function, procedure, procedural language, schema, tablespace

For Example: GRANT SELECT ON employee TO jack;

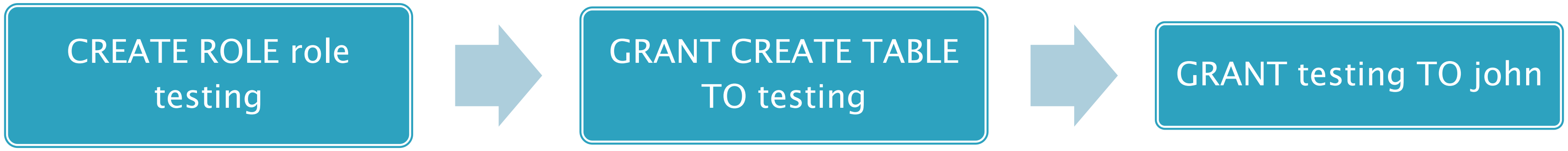
SQL – Command

REVOKE

The REVOKE command removes user access rights or privileges to the database objects.

```
REVOKE
   privilege_name
ON
   object_name
FROM
   {user_name |PUBLIC |role_name}
```

ROLE



CREATE ROLE role testing

GRANT CREATE TABLE TO testing

GRANT testing TO john

REVOKE CREATE TABLE FROM testing;

DROP ROLE testing;

TCL Commands

Transaction **C**ontrol **L**anguage

Command	Description
COMMIT	Perform/Execute a transaction (apply the operations on the objects) All changes made by the transaction become visible to others and are guaranteed to be durable
ROLLBACK	Rollbacks a transaction in case of any error occurs (restore to a previously defined state to recover)
SAVEPOINT	Sets a savepoint/restore point within a transaction (selectively discard parts of a transaction)
SET TRANSACTION	Specify characteristics for the transaction (isolation level access mode (R/W or R/O), and the deferrable mode)

SQL – Part 1 – Commands

Transaction

When you execute the following INSERT statement:

```
INSERT INTO accounts(name,balance) VALUES('Bob',10000);
```

A new row is inserted into the accounts table immediately.

In this case, you did not know when the transaction began and had no chance to intercept the change such as undoing it if something goes wrong.

How to garantee the ACID properties of a transaction ?

SQL – Part 1 – Commands

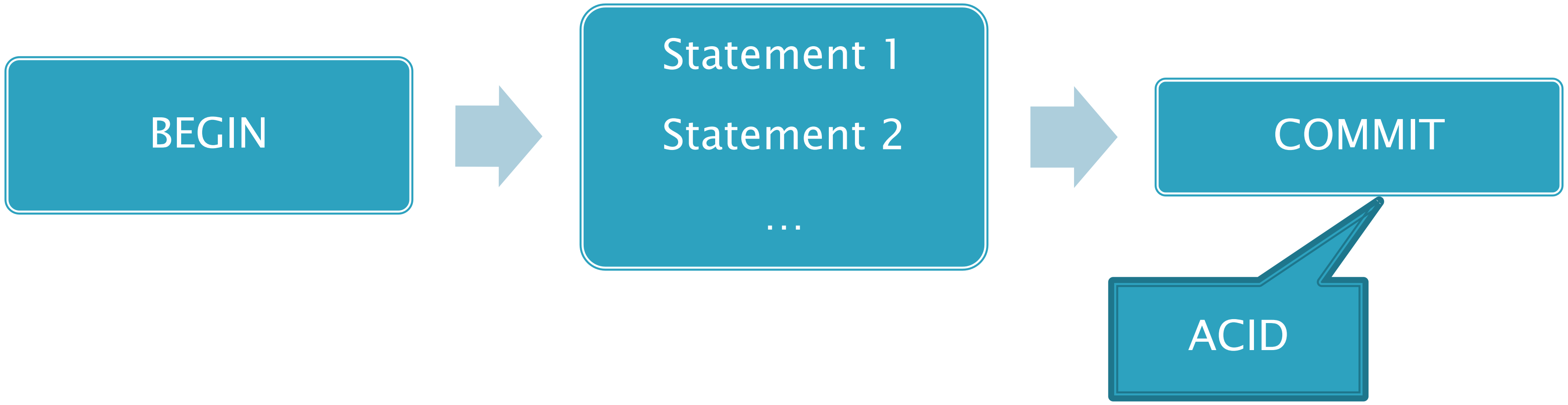
Transaction

A database transaction is a single unit of work which may consist of one or more operations accomplished in a logical order.

Each transaction begins with a specific task and ends when all the tasks in the group successfully complete.

If any of the tasks fail, the transaction fails.

Therefore, a transaction has only two results: SUCCESS or FAILURE.



SQL – Command

COMMIT

is used to save changes invoked by a transaction to the database.

All changes made by the transaction become visible to others and are guaranteed to be durable if a crash occurs.

To commit the current transaction and make all changes permanent:

```
COMMIT;
```

Example:

```
BEGIN;
UPDATE accounts
SET balance = balance + 1000
WHERE id = 2;
COMMIT;
```

Note: The SQL standard specifies two forms COMMIT and COMMIT WORK.

✋ AUTOCOMMIT Session Parameter

SQL – Command

ROLLBACK

aborts the current transaction.

This command rolls back the current transaction and causes all the updates made by the transaction to be discarded.

To abort all changes:

```
ROLLBACK;
```

Example:

```
BEGIN;
UPDATE accounts
SET balance = balance + 1000
WHERE id = 2;
ROLLBACK;
```

Note: The SQL standard specifies two forms ROLLBACK and ROLLBACK WORK

SQL – Command

SAVEPOINT

defines a new savepoint within the current transaction.

A savepoint is a special mark inside a transaction that allows all commands that are executed after it was established to be rolled back, restoring the transaction state to what it was at the time of the savepoint.

To establish a savepoint:

```
BEGIN;
    INSERT INTO table1 VALUES (1);
    SAVEPOINT my_savepoint;
    INSERT INTO table1 VALUES (2);
    ROLLBACK TO SAVEPOINT my_savepoint;
    INSERT INTO table1 VALUES (3);
    RELEASE SAVEPOINT my_savepoint;
COMMIT;
```

SQL – Command

SET TRANSACTION

set the characteristics of the current transaction

This command sets the characteristics of the current transaction.

The available transaction characteristics are:

- the transaction isolation level
- the transaction access mode (read/write or read-only)
- *the deferrable mode*

Example :

```
SET TRANSACTION  ISOLATION LEVEL SERIALIZABLE;
```

Note :

These commands are defined in the SQL standard, except for the DEFERRABLE transaction mode.

In the SQL standard, there is one other transaction characteristic that can be set with these commands: the size of the diagnostics area.

This concept is specific to embedded SQL, and therefore is not implemented in standard RDBMS,

SQL – Command

SET TRANSACTION

The transaction isolation level

```
{ SERIALIZABLE | REPEATABLE READ | READ COMMITTED | READ UNCOMMITTED }
```

READ COMMITTED
A statement can only see rows committed before it began. This is the default.

REPEATABLE READ
All statements of the current transaction can only see rows committed before the first query or data-modification statement was executed in this transaction.

SERIALIZABLE
All statements of the current transaction can only see rows committed before the first query or data-modification statement was executed in this transaction. If a pattern of reads and writes among concurrent serializable transactions would create a situation which could not have occurred for any serial (one-at-a-time) execution of those transactions, one of them will be rolled back with a serialization_failure error.

Note:The transaction isolation level cannot be changed after the first query or data-modification statement (SELECT, INSERT, DELETE, UPDATE, FETCH, or COPY) of a transaction has been executed.

SQL – Command

SET TRANSACTION

The transaction access mode

```
{ READ WRITE | READ ONLY }
```

The transaction access mode determines whether the transaction is read/write or read-only. Read/write is the default. When a transaction is read-only, the following SQL commands are disallowed: INSERT, UPDATE, DELETE, and COPY FROM if the table they would write to is not a temporary table; all CREATE, ALTER, and DROP commands; COMMENT, GRANT, REVOKE, TRUNCATE; and EXPLAIN ANALYZE and EXECUTE if the command they would execute is among those listed. This is a high-level notion of read-only that does not prevent all writes to disk.

SQL – Command

SET TRANSACTION

The deferrable mode

```
{ [NOT] DEFERRABLE }
```

The DEFERRABLE transaction property has no effect unless the transaction is also SERIALIZABLE and READ ONLY. When all three of these properties are selected for a transaction, the transaction may block when first acquiring its snapshot, after which it is able to run without the normal overhead of a SERIALIZABLE transaction and without any risk of contributing to or being canceled by a serialization failure. This mode is well suited for long-running reports or backups.

SQL – Advanced Types

Integer

Attempts to store values outside of the allowed range will result in an error.

Numeric (/ Decimal)

The type numeric can store numbers with a very large number of digits. It is especially recommended for storing monetary amounts and other quantities where exactness is required.

In addition to ordinary numeric values, the numeric type allows the special value NaN, meaning “not-a-number”. Any operation on NaN yields another NaN.

When rounding values, the numeric type rounds ties away from zero, while the real and double precision types round ties to the nearest even number (depending on CPU architecture)

SQL – Advanced Types

Real / Double Précision

Behaviors depend on underlying processor, operating system, and compiler.

If you require exact storage and calculations (such as for monetary amounts), use the numeric type instead.

Comparing two floating-point values for equality might not always work as expected.

Values that are too large or too small will cause an error:

- Rounding might take place if the precision of an input number is too high.
- Numbers too close to zero that are not representable as distinct from zero will cause an underflow error.

In addition to ordinary numeric values, the floating-point types have several special values:

- Infinity
- -Infinity
- NaN

IEEE754 specifies that NaN should not compare equal to any other floating-point value. PostgreSQL treats NaN values as equal, and greater than all non-NaN values.

SQL – Advanced Types

Serial (/ smallserial / bigserial)

A NOT NULL constraint is applied to ensure that a null value cannot be inserted.

You would also want to attach a UNIQUE or PRIMARY KEY constraint to prevent duplicate values from being inserted by accident.

To insert the next value, this can be done either by excluding the column from the list of columns in the INSERT statement, or through the use of the DEFAULT key word.

Another way is to use the SQL-standard identity column feature, described at CREATE TABLE ()

Date/Time Input

https://www.postgresql.org/docs/current/datatype-datetime.html

SQL – Advanced Types

Enumerated Types

The ordering of the values in an enum type is the order in which the values were listed when the type was created. All standard comparison operators and related aggregate functions are supported for enums.

Enum labels are case sensitive, so 'happy' is not the same as 'HAPPY'. White space in the labels is significant too.

Each enumerated data type is separated and cannot be compared with other enumerated types

BLOB

The “hex” format encodes binary data as 2 hexadecimal digits per byte, most significant nibble first. The entire string is preceded by the sequence \x

A binary string is a sequence of octets (or bytes). Binary strings are distinguished from character strings. Binary strings specifically allow storing octets of value zero and other “non-printable” octets (usually, octets outside the decimal range 32 to 126).

SQL – Advanced Types

User-defined Data Type (UDT)

The UDT introduced on SQL:1999.

Extended on SQL:2011.

UDT is a data type that derived from an existing data type.

You can use UDTs to extend the built-in types already available and create your own customized data types.

2 Types of UDT:

- Distinct Type

A distinct type is a UDT that shares its internal representation with an existing built-in data type (its "source" type).

- Structured Type

A structured type is a UDT that has a structure that is defined in the database. It contains a sequence of named attributes, each of which has a data type.

SQL – Advanced Types

User defined type (UDT)

Distinct Type

For Example:

```
CREATE DISTINCT TYPE DollarUS AS DECIMAL (9,2);
```

Then

```
CREATE TABLE orders (
  OrderID INTEGER PRIMARY KEY,
  CustomerID INTEGER,
  NetAmount DollarUS,
  Tax DollarUS,
  TotalAmount DollarUS,
);
```

SQL – Advanced Types

User defined type (UDT)

Structured Type

For Example:

To create:

```
CREATE TYPE film_summary AS (
    film_id INT,
    title VARCHAR,
    release_year YEAR
);
```

To insert:

```
INSERT INTO table VALUES (ROW(v1,'t1',y1), ...);
```

To acces:

```
SELECT (col).title FROM table WHERE (col).release_year > ‘2000’;
```

SQL – Advanced Types

Domain

- A domain is a data type with optional constraints e.g., NOT NULL, CHECK…
- A domain has a unique name within the schema scope.
- A domain is useful for centralizing management of fields with common constraints.

For example, some tables might contain the text columns that require a CHECK constraint to ensure values are not null and also do not contain space.

```
CREATE DOMAIN contact AS
    VARCHAR NOT NULL CHECK (value !~ '\s');
```

```
CREATE TABLE mail_list (
    id serial PRIMARY KEY,
    first_name contact,
    last_name contact,
    email VARCHAR NOT NULL
);
```

SQL – Advanced Types

Sequences

A sequence is a special kind of database object that generates a sequence of integers.

This involves creating and initializing a new special single-row table

```
CREATE [ TEMPORARY | TEMP ] SEQUENCE [ IF NOT EXISTS ] name
  [ INCREMENT [ BY ] increment ]
  [ MINVALUE minvalue | NO MINVALUE ]
  [ MAXVALUE maxvalue | NO MAXVALUE ]
  [ START [ WITH ] start ]
```

After a sequence is created, functions nextval, currval, and setval are used to operate on the sequence.

You can use a query like:

```
SELECT * FROM name;
```

to examine the parameters and current state of a sequence.

Use DROP SEQUENCE to remove a sequence.

Sequences / SERIAL

PostgreSQL SERIAL pseudo-type is used to create an auto-increment column in a database table.

It is a sequence used as a primary key column.

When creating a new table:

```
CREATE TABLE table_name(
    id SERIAL,
    …
);
```

PostgreSQL will perform the following:

- Creates a sequence object and set the next value generated by the sequence as the default value for the column.
- Adds the NOT NULL constraint to the column because a sequence always generates an integer, which is a non-null value.
- Assigns the owner of the sequence to the id column; as a result, the sequence object is deleted when the id column or table is dropped

AUTO_INCREMENT MySQL®

SQL – NULL

NULL is a state not a value

Because Null is not a data value, but a marker for an absent value, using mathematical operators on Null gives an unknown result

```
10 x NULL                    ⇒ NULL
'Fish ' || NULL || 'Chips‘   ⇒ NULL
SELECT 10 = NULL             ⇒ Unknown
```

Null can return values if the absent value is not relevant to the outcome of the operation

```
SELECT NULL OR TRUE          ⇒ TRUE
```

Common mistakes

```
SELECT *                     ⇒ ZERO
FROM t                         ROWS
WHERE i = NULL;
```

comparison of the i column with Null always returns Unknown

```
SELECT *
FROM sometable
WHERE num <> 1;
```

Rows where num is NULL will not be returned

```
SELECT *
FROM sometable
WHERE LENGTH(string) < 20;
```

Rows where string is NULL will not be returned.

SQL – WITH

The WITH Clause

- Provides a way to write auxiliary statements for use in a larger query.
- Helps in breaking down complicated and large queries into simpler forms, which are easily readable.
- Can be thought of as defining temporary tables that exist just for one query.

set of columns

```
WITH
  temp_name_1 AS (
    SELECT statement
    FROM tables
    WHERE conditions)
SELECT columns
  FROM table2
  WHERE columns in (
    SELECT columns
    FROM temp_name_1)
  [ORDER BY columns];
```

These statements often referred to as Common Table Expressions or CTEs

SQL – Joining Data

- Joins
- Union
- Intersect
- Except

SQL – Joining Data - JOINS

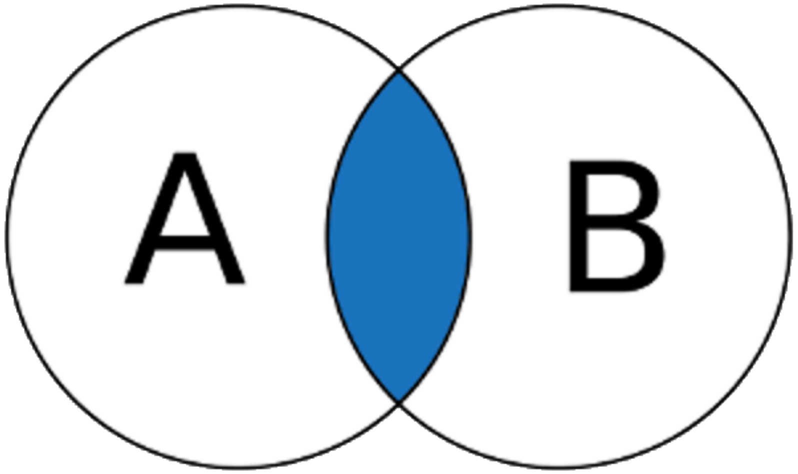
We will show you a brief overview of joins.

- Inner Join – selects rows from one table that have the corresponding rows in other tables.
- Left Join – selects rows from one table that may or may not have the corresponding rows in other tables.
- Self-join – joins a table to itself by comparing a table to itself.
- Full Outer Join – returns all records when there is a match in either left table or right table records.
- Cross Join – produces a Cartesian product of the rows in two or more tables.
- Natural Join – joins two or more tables using implicit join condition based on the common column names in the joined tables.

SQL – JOIN (inner)

Inner Join

selects rows from one table that have the corresponding rows in other tables.



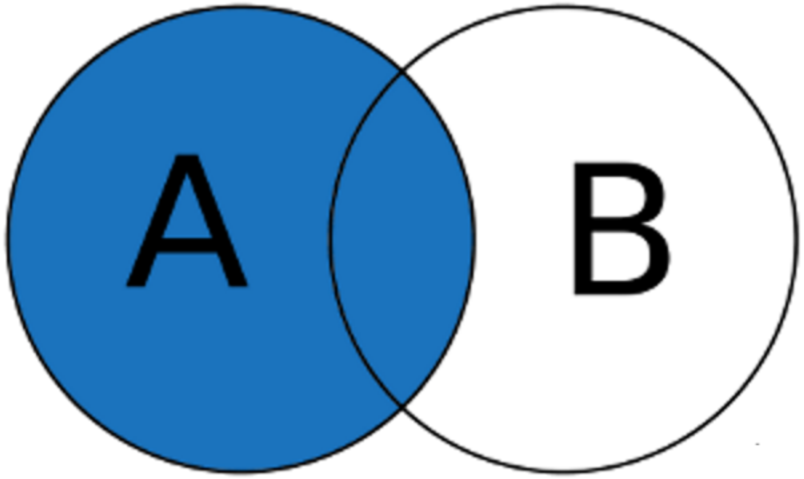
```
SELECT *
FROM A
INNER JOIN B ON A.key = B.key
```

The INNER JOIN keyword selects all rows from both tables as long as there is a match between the columns. If there are records in the "Orders" table that do not have matches in "Customers", these orders will not be shown!

SQL – JOIN (left)

Left Join

returns all rows from the left table
that may or may not have the corresponding rows in other tables.
The result is NULL from the right side, if there is no match.



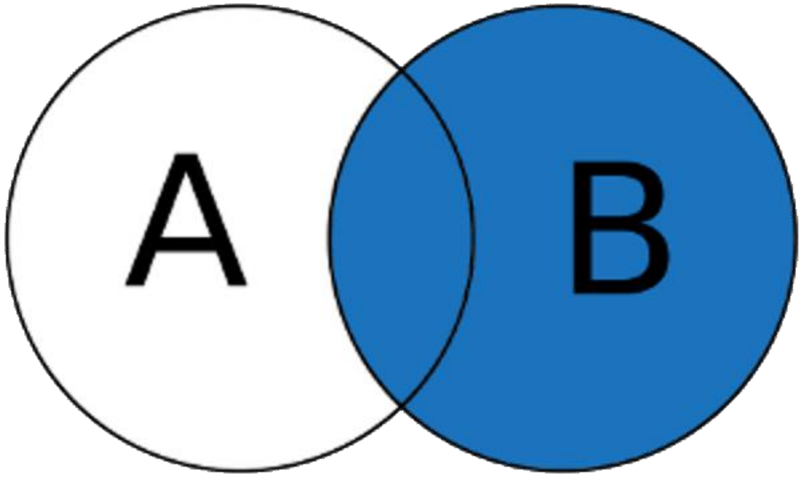
```
SELECT *
FROM A
LEFT JOIN B ON A.key = B.key
```

Note: In some databases LEFT JOIN is called LEFT OUTER JOIN.

SQL – JOIN (right)

Right Join

returns all rows from the right table
that may or may not have the corresponding rows in other tables.
The result is NULL from the left side, if there is no match.



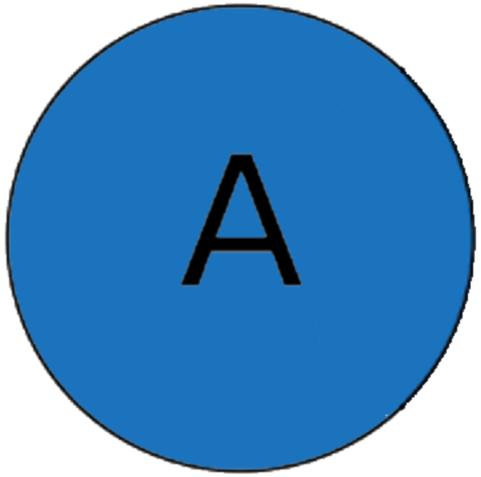
```
SELECT *
FROM A
RIGHT JOIN B ON A.key = B.key
```

Note: In some databases LEFT JOIN is called RIGHT OUTER JOIN.

SQL – JOIN (self)

Self-join

joins a table to itself. Self-joins are useful for comparing values in a column of rows within the same table.



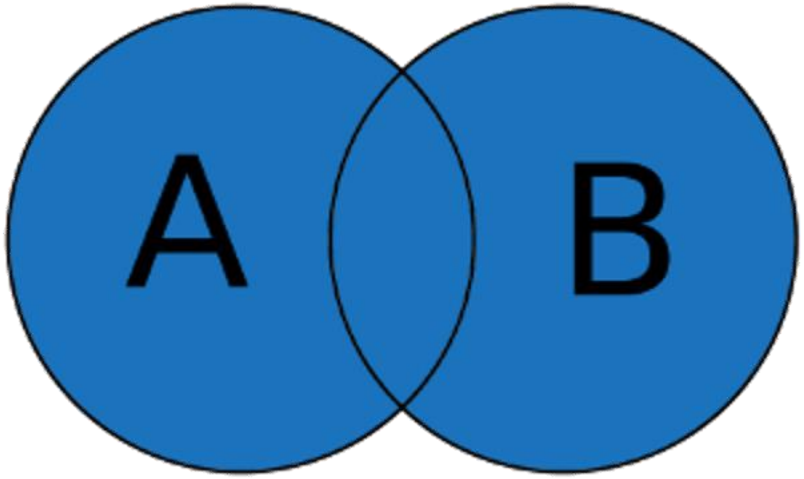
```
SELECT *
FROM A a1
INNER JOIN A a2 ON a1.key = a2.key
```

Note : You can also use the LEFT JOIN or RIGHT JOIN clause instead of INNER JOIN

SQL – JOIN (full outer)

Full Outer Join

returns all records when there is a match in either left table or right table records.



```
SELECT *
FROM A
FULL OUTER JOIN B
ON A.key = B.key
```

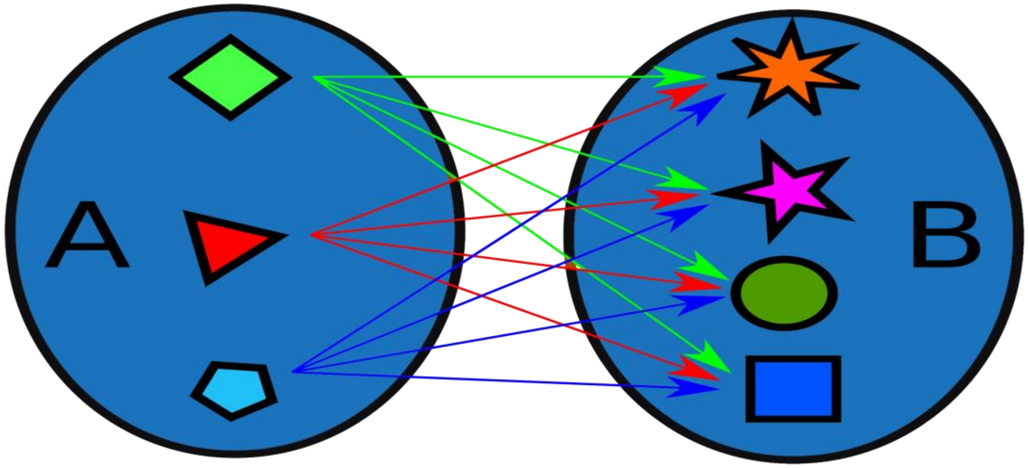
Note: FULL OUTER JOIN can potentially return very large result-sets!

Cross Join

produces the Cartesian Product of rows in two or more tables.

Each row from the 1st table is combined with each row from the 2nd one.

Different from the other JOIN operators, the CROSS JOIN does not have any matching condition in the join clause.



```
SELECT *
FROM A
CROSS JOIN B
```

```
SELECT *
FROM A, B
```

⇔

```
SELECT *
FROM A
INNER JOIN B
ON TRUE;
```

Note: If T1 has N rows, T2 has M rows, the result set will have N x M rows.

SQL – JOIN (natural)

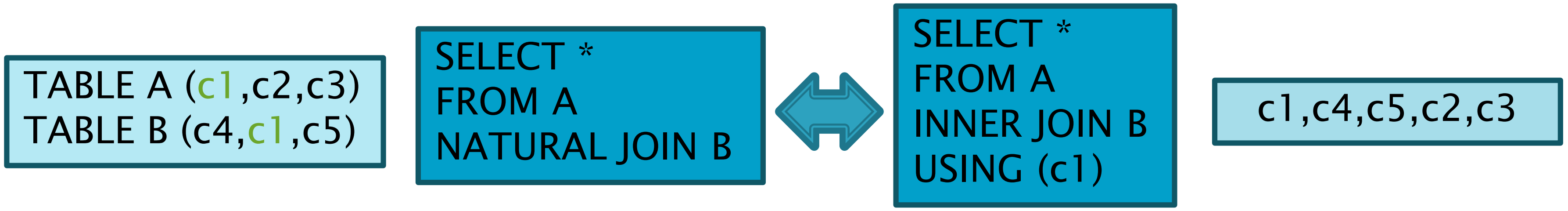
Natural Join

creates an implicit join based on the same column names in the joined tables.

A natural join can be an inner join, left join, or right join. If you do not specify a join explicitly, the INNER JOIN is used by default.

If asterisk (*) is used in the select list, the result will contain the following columns:

- All the common columns, which are the columns in the both tables that have the same name
- Every column in the first and second tables that is not a common column



```
TABLE A (c1,c2,c3)
TABLE B (c4,c1,c5)
```

```
SELECT *
FROM A
NATURAL JOIN B
```

```
SELECT *
FROM A
INNER JOIN B
USING (c1)
```

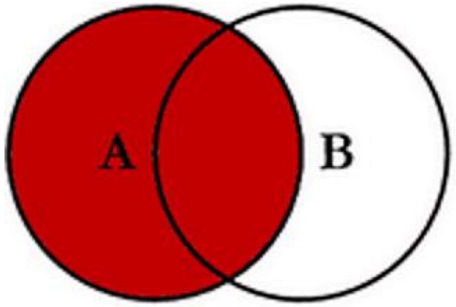
```
c1,c4,c5,c2,c3
```

Note: The convenience is that it does not require you to specify the join clause because it uses an implicit join clause based on the common column.

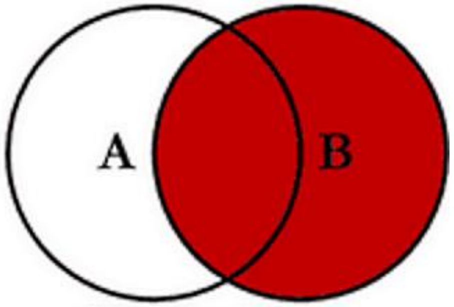
Avoid using it whenever possible because it may cause an unexpected result

SQL – Joining multiple tables

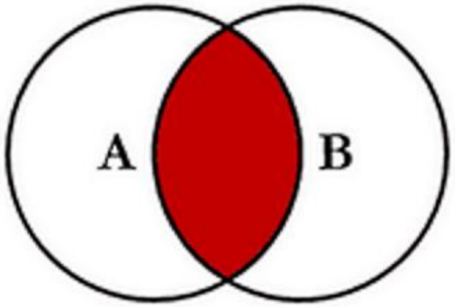
CHEATSHEET
SQL
JOINS



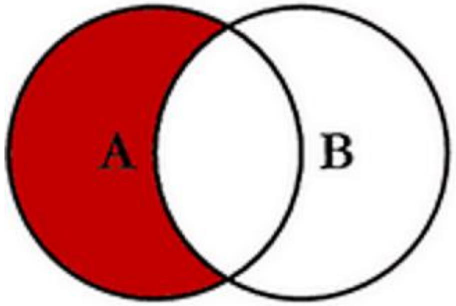
```
SELECT <select_list>
FROM TableA A
LEFT JOIN TableB B
ON A.Key = B.Key
```



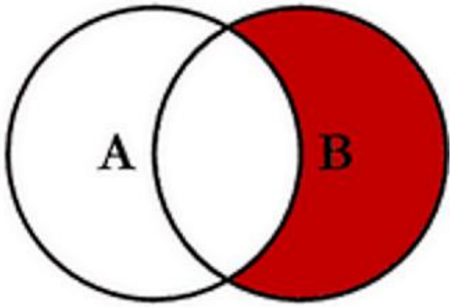
```
SELECT <select_list>
FROM TableA A
RIGHT JOIN TableB B
ON A.Key = B.Key
```



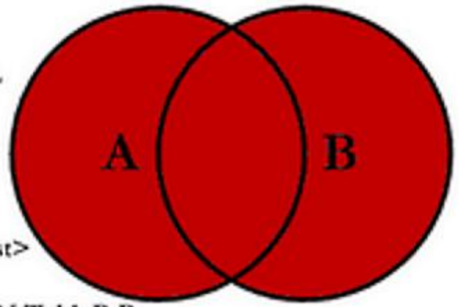
```
SELECT <select_list>
FROM TableA A
INNER JOIN TableB B
ON A.Key = B.Key
```



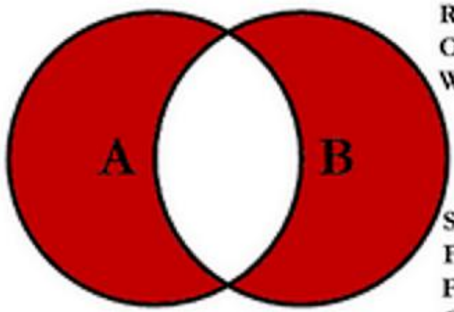
```
SELECT <select_list>
FROM TableA A
LEFT JOIN TableB B
ON A.Key = B.Key
WHERE B.Key IS NULL
```



```
SELECT <select_list>
FROM TableA A
RIGHT JOIN TableB B
ON A.Key = B.Key
WHERE A.Key IS NULL
```



```
SELECT <select_list>
FROM TableA A
FULL OUTER JOIN TableB B
ON A.Key = B.Key
```



```
SELECT <select_list>
FROM TableA A
FULL OUTER JOIN TableB B
ON A.Key = B.Key
WHERE A.Key IS NULL
OR B.Key IS NULL
```

SQL – Joining Data - UNION

The UNION operator combines result sets of two or more SELECT statements into a single result set.

The following are rules applied to the queries:

- Both queries must return the same number of columns.
- The corresponding columns in the queries must have compatible data types.

removes all duplicate rows

  Unless the UNION ALL is used.

No order

  To sort the rows in the combined result set by a specified column, you use the ORDER BY clause.

```
SELECT
   column_list
FROM
   tbl_name_1
UNION
SELECT
   column_list
FROM
   tbl_name_2
[ORDER BY
   <clause>]
```

SQL – Joining Data - INTERSECT

The INTERSECT operator combines the result sets of two or more SELECT statements into a single result set.
The INTERSECT operator returns any rows that are available in both result set or returned by both queries

The following are rules applied to the queries:
- The number of columns and their order in the SELECT clauses must be the same.
- The data types of the columns must be compatible

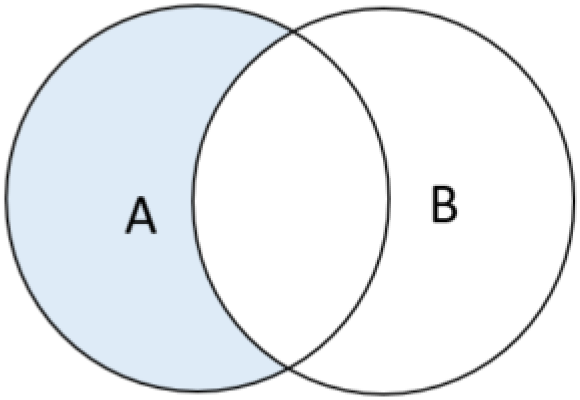
To sort the result set returned by the INTERSECT operator, place the ORDER BY clause at the end of the statement

```
SELECT
   column_list
FROM
   tbl_name_1
INTERSECT
SELECT
   column_list
FROM
   tbl_name_2
[ORDER BY
   <clause>]
```

SQL – Joining Data - EXCEPT

the SQL EXCEPT operator returns the rows in the first query that do not appears in the output of the second query.
It returns rows by comparing the result sets of two or more queries.

To combine the queries using the EXCEPT operator:
- The number of columns & their orders must be the same in the 2 queries.
- The data types of the respective columns must be compatible.



```
SELECT
   column_list
FROM
   tbl_name_1
EXCEPT
SELECT
   column_list
FROM
   tbl_name_2
[ORDER BY
   <clause>]
```

To sort the result set returned by the EXCEPT operator, place the ORDER BY clause at the end of the statement.

MINUS ORACLE MySQL

SQL – JOIN Example Tables

TABLE: lp_cust

custId	firstName	lastName	userName
1	'Fred'	'Flinstone'	'freddo'
2	'John'	'Smith'	'jsmith'
3	'Homer'	'Simpson'	'homey'
4	'Homer'	'Brown'	'notsofamous'
5	'Ozzy'	'Ozzbourn'	'sabbath'
6	'Homer'	'Gain'	'noplacelike'
7	'Jack'	'Brown'	'bigjack'
8	'Morten'	'Harket'	'aha'
9	'John'	'Smith'	'jsmith1'

TABLE: loyalty_program

1	'gold'
2	'silver'
3	'silver'
4	'bronze'
5	'bronze'
10	'silver'
10	'bronze'
11	'gold'

TABLE: lp_actor

actorId	firstName	lastName
1	'Fred'	'Flinstone'
2	'Homer'	'Simpson'
3	'Greta'	'Garbo'
4	'Jennifer'	'Lawrence'
5	'Will'	'Smith'

SQL – Grouping Sets

the GROUPING SETS OPERATOR generates a result set equivalent to which generated by the UNION ALL of the multiple GROUP BY clauses.

A grouping set is a set of columns by which you group.

```
SELECT
   c1,c2,aggr_fct(c3)
FROM
   t1
GROUP BY c1,c2
```

```
SELECT
   c1, aggr_fct(c3)
FROM
   t1
GROUP BY c1
```

```
SELECT
   c2, aggr_fct(c3)
FROM
   t1
GROUP BY c2
```

```
SELECT
    c1,c2,aggr_fct(c3)
FROM
    t1
GROUP BY c1,c2

UNION ALL

SELECT
    c1, NULL, aggr_fct(c3)
FROM
    t1
GROUP BY c1

UNION ALL

SELECT
    NULL, c2, aggr_fct(c3)
FROM
    t1
GROUP BY c2
```

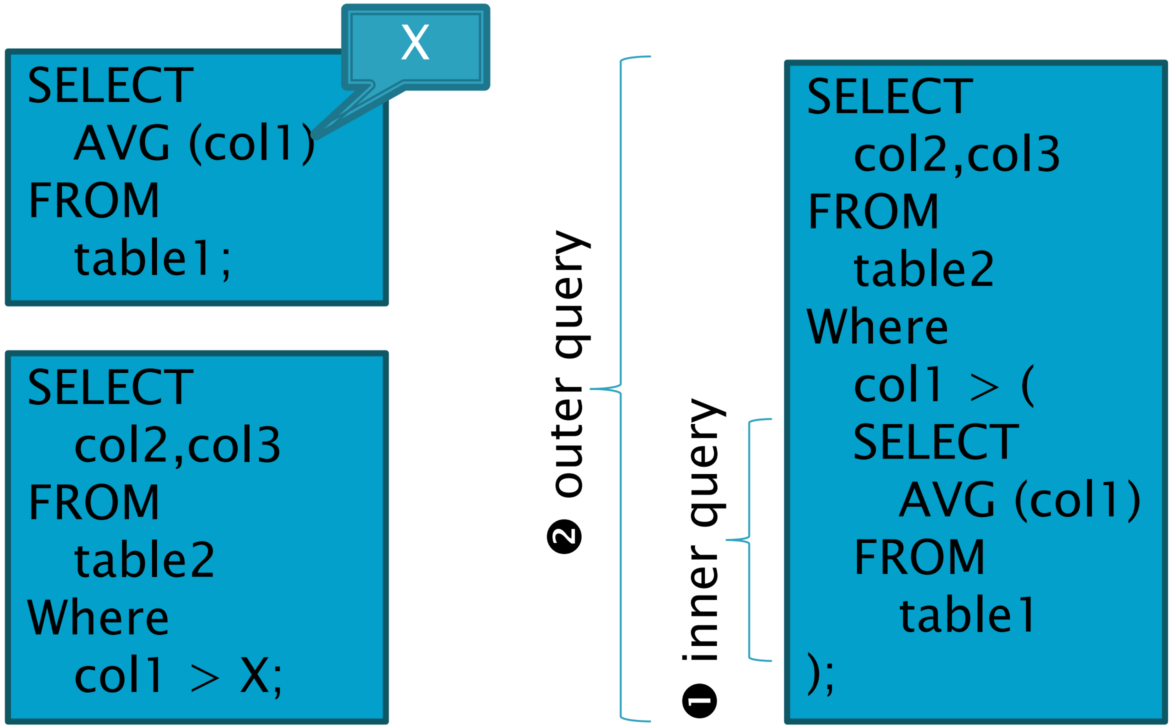
```
SELECT
    c1,
    c2,
    aggr_fct(c3)
FROM
    t1
GROUP BY
    GROUPING SETS (
        (c1, c2),
        (c1),
        (c2)
)
```

- Quite lengthy
- Performance issue
  (multiple scans of t1)

SQL – SubQueries

The SQL subquery mechanism allows you to construct complex queries.

A subquery is a query nested inside another query such as SELECT, INSERT, DELETE and UPDATE.
To construct a subquery, the second query is put in brackets and used in the WHERE clause as an expression.



```
SELECT
   AVG (col1)
FROM
   table1;
```

```
SELECT
   col2,col3
FROM
   table2
Where
   col1 > X;
```

```
SELECT
   col2,col3
FROM
   table2
Where
   col1 > (
   SELECT
      AVG (col1)
   FROM
      table1
);
```

SQL – SubQueries

A subquery can return zero or more rows.
To use this subquery, add the IN operator in the WHERE clause.

```
SELECT
   col2,col3
FROM
   table2
Where
   col1 in (
   SELECT
       col4
   FROM
       table1
   WHERE condition
);
```

SQL – Part 1 – Conclusion

```
SELECT questions FROM audience;
```