Database Management System – EPAM

What is Information?

Data

Data is a set of facts in a “raw”, non-processed state. Data requires processing to become useful for people.

Minsk, Ivan, 7, Kuprevicha, 80221234567, 220022, Ivanov

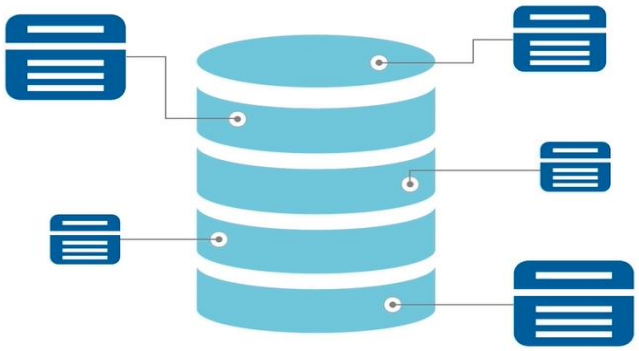
processing

Information

When data is processed, organized, structured and can be used by a human or a PC to gain value / profit.

Ivan Ivanov
Kuprevicha, 7
Minsk, 220022
8-(022)-1234567

What is DataBase (DB)?



Real life examples:

- Booking tickets
- Weather prediction
- Supermarket purchases
- Financial analytics
- Instagram
- Training courses

Common DataBase Types

Trainees
Name
Date of Birth
Gender
Education
Prog. experience

Columns

Table

Trainees				
Name	**Date of Birth**	**Gender**	**Education**	**Prog. experience**
Maria Petrova	14.03.1988	F	Completed High	False
Ivan Ivanov	25.05.1979	M	Incomplete High	True
...	...	...	...	...

Rows

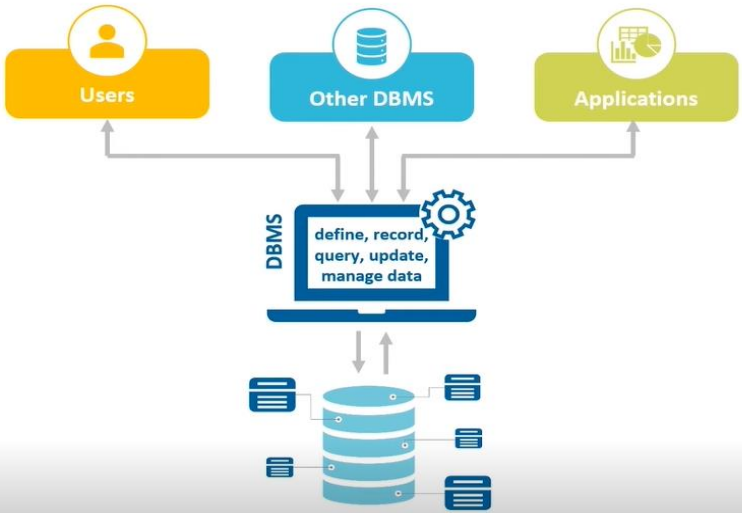
Cell

Common DataBase Types

Relational database	Key-value database
Table	Collection
Table row	Element of collection
Table column	Key of an element
Table cell	Value by element's key

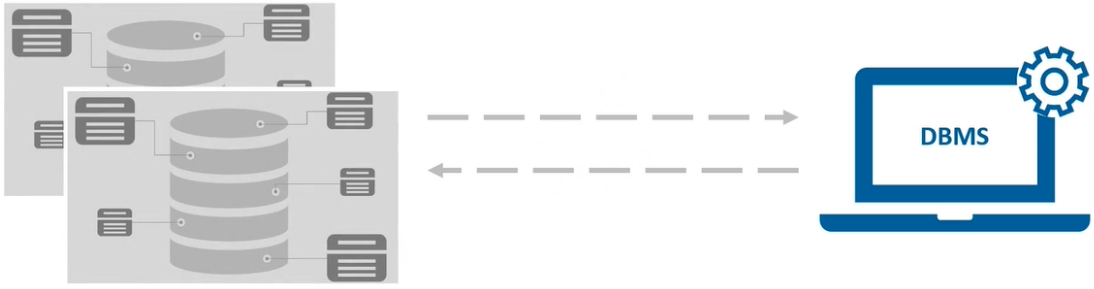

```
"Trainees": [{
      "Name":              "Maria Petrova",
      "Date of Birth":     "14.03.1988",
      "Gender":            "F",
      "Education":         "Completed High",
      "Prog. experience": "False"
   }, {
      "Name":              "Ivan Ivanov",
      "Date of Birth":     "25.05.1979",
      "Gender":            "M",
      "Education":         "Incomplete High",
      "Prog. experience": "True"
   }
]
```


What is DataBase Management System (DBMS)



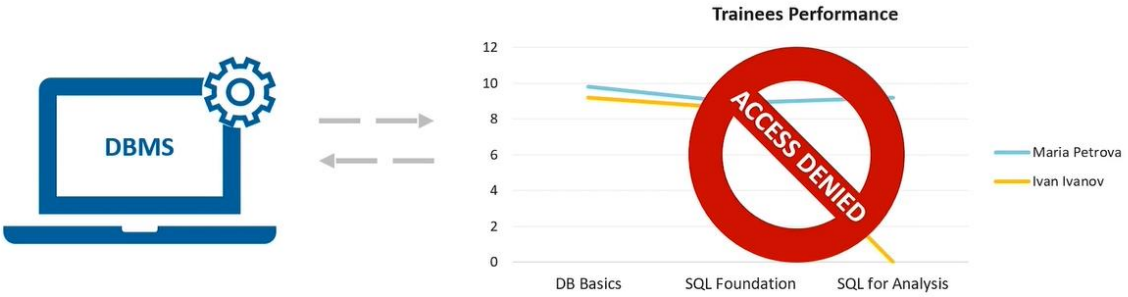
What RDBMS is for?

Manage database backup and recovery processes



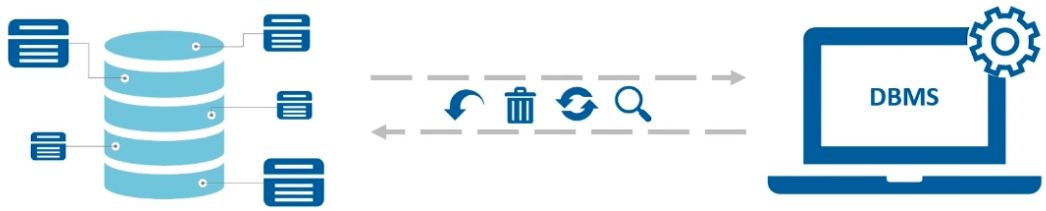
What RDBMS is for?

Restrict data access according to predefined rules



What RDBMS is for?

Allow database access via one of predefined interfaces (most common is SQL query)



What RDBMS is for?

Support data consistency when multiple users work with same pieces of data

Relational DataBase Components: Table

Name	Date of Birth	Gender	Education	Prog. experience
Maria Petrova	14.03.1988	F	Completed High	False
Ivan Ivanov	25.05.1979	M	Incomplete High	True
...	...	...	...	...

text

Relational DataBase Components: Table

Name	Date of Birth	Gender	Education	Prog. experience
Maria Petrova	14.03.1988	F	Completed High	False
Ivan Ivanov	25.05.1979	M	Incomplete High	True
...	...	...	...	...

boolean

Relational DataBase Components: Constraint

Name	Date of Birth	Gender	Education	Prog. experience
Maria Petrova	14.03.1988	F	Completed High	False
Ivan Ivanov	25.05.1979	M	Incomplete High	True
Petr Pirogov	17.11.1987	M	Super High	True
Inna Sidorova	01.01.1992	F	Upper High	False
Volha Hlebava	06.06.1991	F	Completed High	Null
Igor Lesik	15.10.2020	M	Completed High	True

As example: **Current Date = 01.01.2020**

Relational DataBase Components: Keys

Surrogate Primary Key

TraineesID	Name	Date of Birth	Gender	Education	Prog. experience
1	Maria Petrova	14.03.1988	F	Completed High	False
2	Ivan Ivanov	25.05.1979	M	Incomplete High	True
3	Ivan Ivanov	25.05.1979	M	Incomplete High	True

Relational DataBase Components: Keys

Trainees

PassportID	Name	Date of Birth	Gender	Education	Prog. experience
MP1111111	Maria Petrova	14.03.1988	F	Completed High	False
BM3333333	Ivan Ivanov	25.05.1979	M	Incomplete High	True

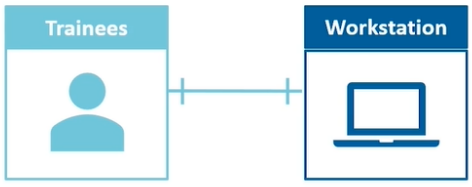
Primary Key

Performance

PassportID	Module	Avg. Mark
MP1111111	DB Basics	9.8
BM3333333	DB Basics	9.2
MP1111111	SQL Foundation	8.9
BM3333333	SQL Foundation	8.6
MP1111111	SQL For Analysis	9.2

Foreign Key

Relational DataBase Components: relationship 1-to-1



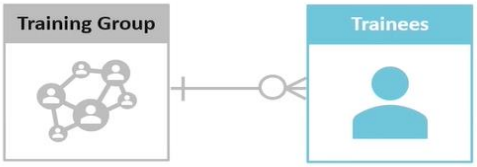
Trainees

PassportID	Name	Date of Birth	Gender	Education	Prog. experience
MP1111111	Maria Petrova	14.03.1988	F	Completed High	False
BM3333333	Ivan Ivanov	25.05.1979	M	Incomplete High	True

Workstation

PassportID	Workstation
MP1111111	CCCCCCC4321
BM3333333	CCCCCCC3322
	CCCCCCC55555

Relational DataBase Components: relationship 1-to-many



Training Group

GroupID	Group Name
1	Data Training 1
2	.Net Training 1

Trainees

GroupID	PassportID	Name	Date of Birth	Gender	Education	Prog. exp
1	MP1111111	Maria Petrova	14.03.1988	F	Completed High	False
1	BM3333333	Ivan Ivanov	25.05.1979	M	Incomplete High	True
2	KH8888888	Marta Yakova	96.06.1984	F	Completed High	True

Relational DataBase Components: relationship many-to-many

Trainees

PassportID	Name
MP1111111	Maria Petrova
BM3333333	Ivan Ivanov

Performance

PassportID	ModuleID	Avg. Mark
MP1111111	1	9.8
MP1111111	2	8.9
MP1111111	3	9.2
BM3333333	1	9.2
BM3333333		

Modules

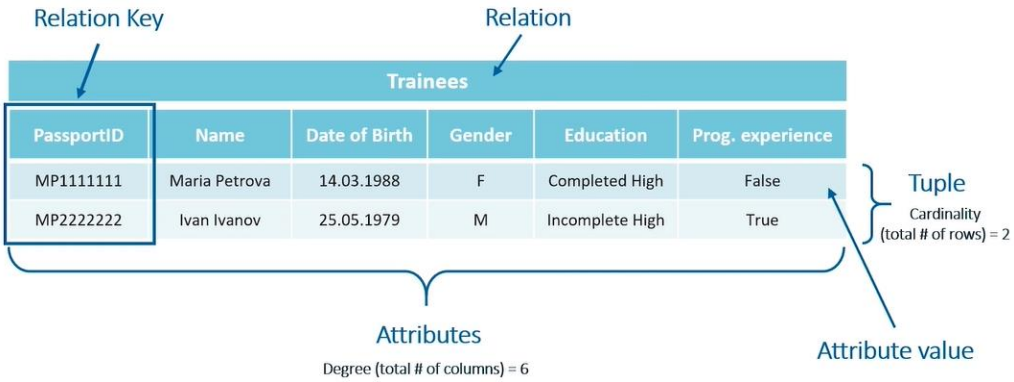
ModuleID	Module Name
1	DB Basics
2	SQL Foundation
3	SQL for Analysis

Relational database components

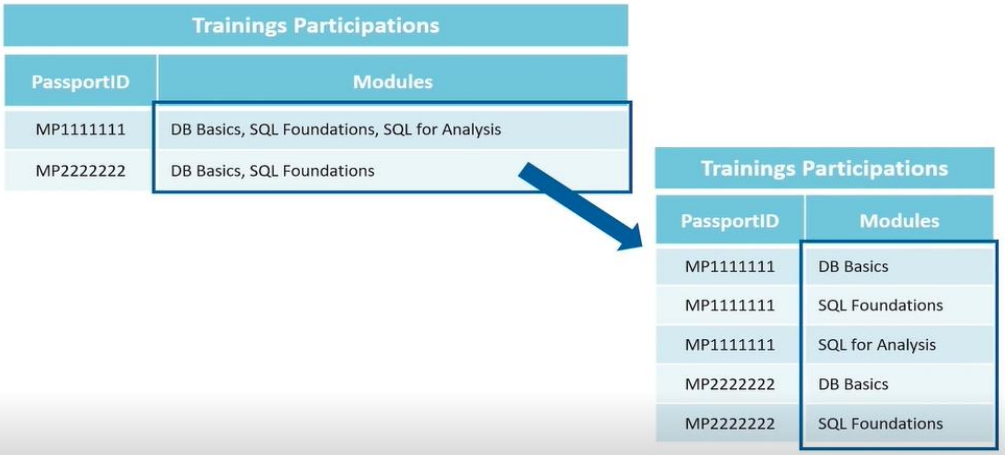
1. Tables
2. Constraints
3. Keys
4. Indexes
5. Views
6. Partitions

- **Database Modelling :**

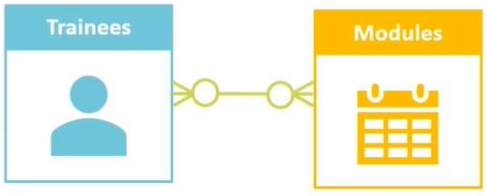
DataBase Structure: Relational Model Terms



DataBase Structure: Rules



DataBase Modeling: Conceptual Model

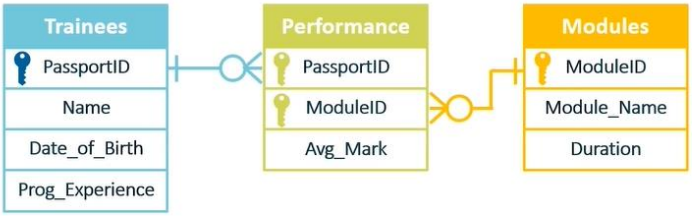


Includes the important entities and the relationships among them

No attributes are specified

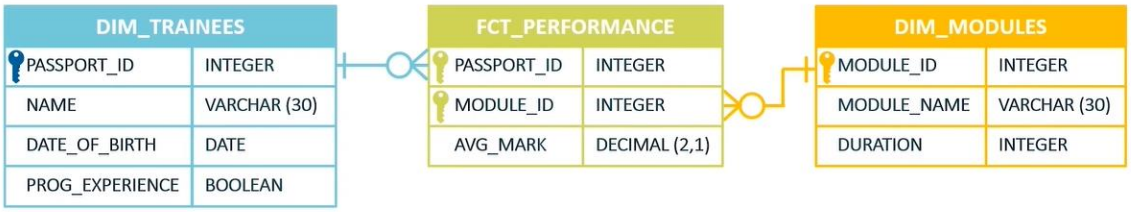
No keys are specified

DataBase Modeling: Logical Model



- All attributes are specified
- The primary key for each entity specified
- Foreign keys are specified
- Relationships are specified
- Normalization occurs at this level

DataBase Modeling: Physical Model



- Convert entities into tables
- Convert attributes into columns
- Convert relationships into foreign keys
- Modify the physical data model based on physical constraints / requirements

DataBase Modeling: Rules

Trainees					
TrainersID	Name	Date_of_Birth	Gender	Education	Prog_Experience
MP1111111	Maria Petrova	14.03.1988	F	Completed High	False
MP2222222	Ivan Ivanov	25.05.1979	M	Incomplete High	True
MP1313311	Petr Pirogov	17.11.1987	M	Completed High	True
BM1112233	Inna Sidorova	01.01.1992	F	Completed High	False

date

Performance		
PassportID	ModuleID	Avg. Mark
MP1111111	1	9.2
MP1111111	2	8.6

integer

decimal

DataBase Structure: Rules

DataBase Tasks

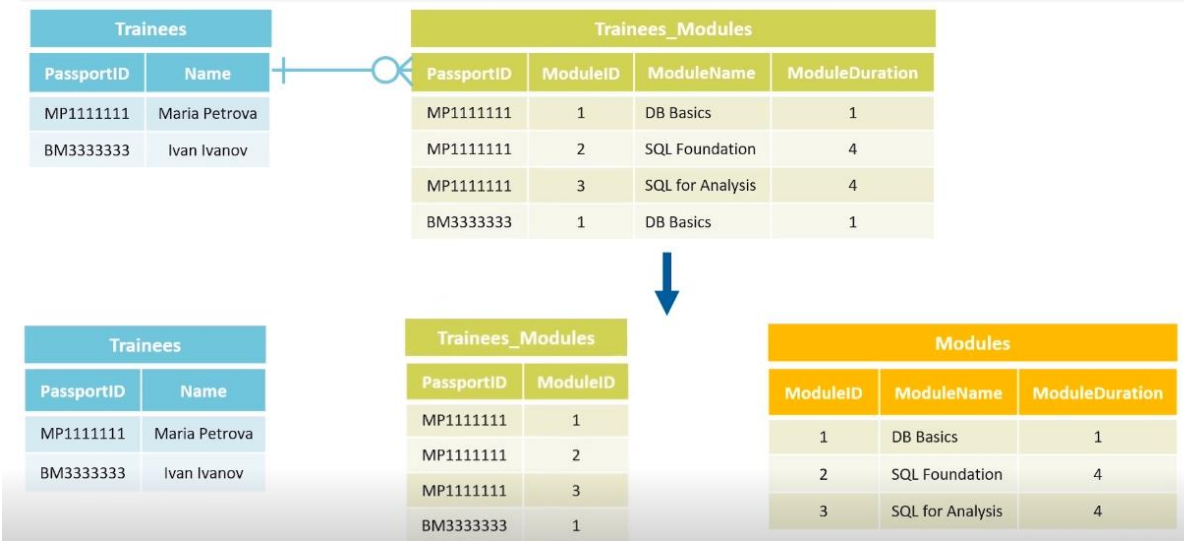
- **Normalization :**

Conversion to 1NF

Trainees						
PassportID	Name	Phone_Number	Date_of_Birth	Gender	Education	Prog_experience
MP1111111	Maria Petrova	375291111111	14.03.1988	F	Completed High	False
BM3333333	Ivan Ivanov	375292222222, 375253333333	25.05.1979	M	Incomplete High	True

Trainees						
PassportID	Name	Phone_Number	Date_of_Birth	Gender	Education	Prog_experience
MP1111111	Maria Petrova	375291111111	14.03.1988	F	Completed High	False
BM3333333	Ivan Ivanov	375292222222	25.05.1979	M	Incomplete High	True
BM3333333	Ivan Ivanov	375253333333	25.05.1979	M	Incomplete High	True

Conversion from 1NF to 2NF



Trainees

PassportID	Name
MP1111111	Maria Petrova
BM3333333	Ivan Ivanov

Trainees_Modules

PassportID	ModuleID	ModuleName	ModuleDuration
MP1111111	1	DB Basics	1
MP1111111	2	SQL Foundation	4
MP1111111	3	SQL for Analysis	4
BM3333333	1	DB Basics	1

Trainees

PassportID	Name
MP1111111	Maria Petrova
BM3333333	Ivan Ivanov

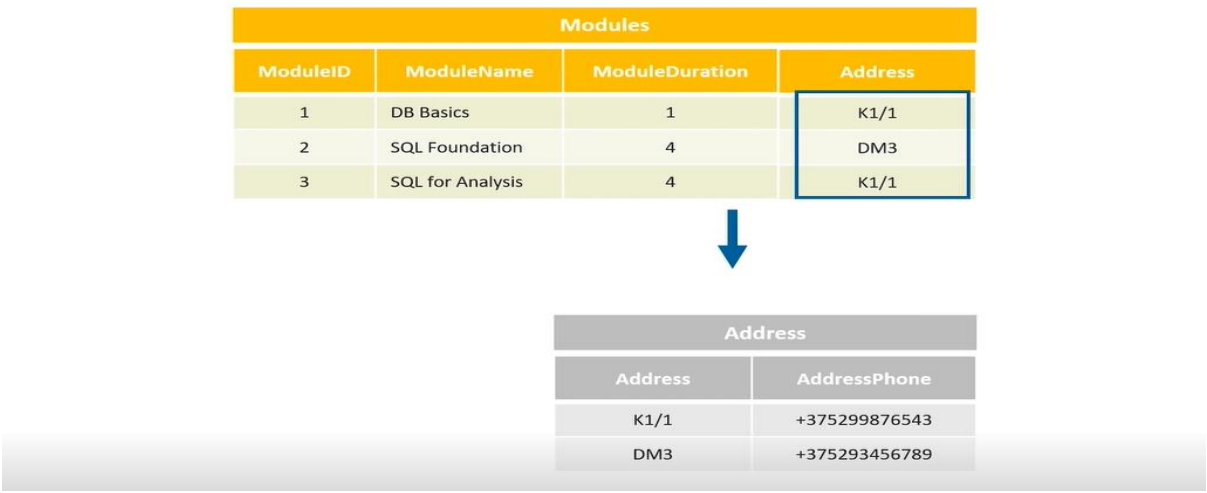
Trainees_Modules

PassportID	ModuleID
MP1111111	1
MP1111111	2
MP1111111	3
BM3333333	1

Modules

ModuleID	ModuleName	ModuleDuration
1	DB Basics	1
2	SQL Foundation	4
3	SQL for Analysis	4

Conversion from 2NF to 3NF



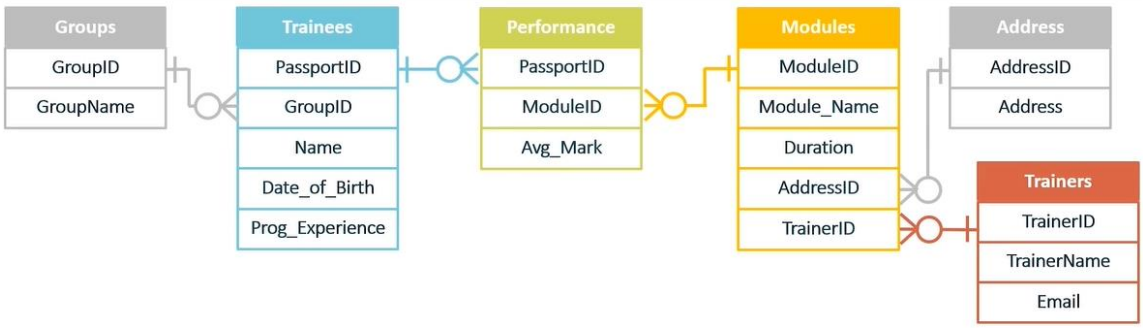
Modules

ModuleID	ModuleName	ModuleDuration	Address
1	DB Basics	1	K1/1
2	SQL Foundation	4	DM3
3	SQL for Analysis	4	K1/1

Address

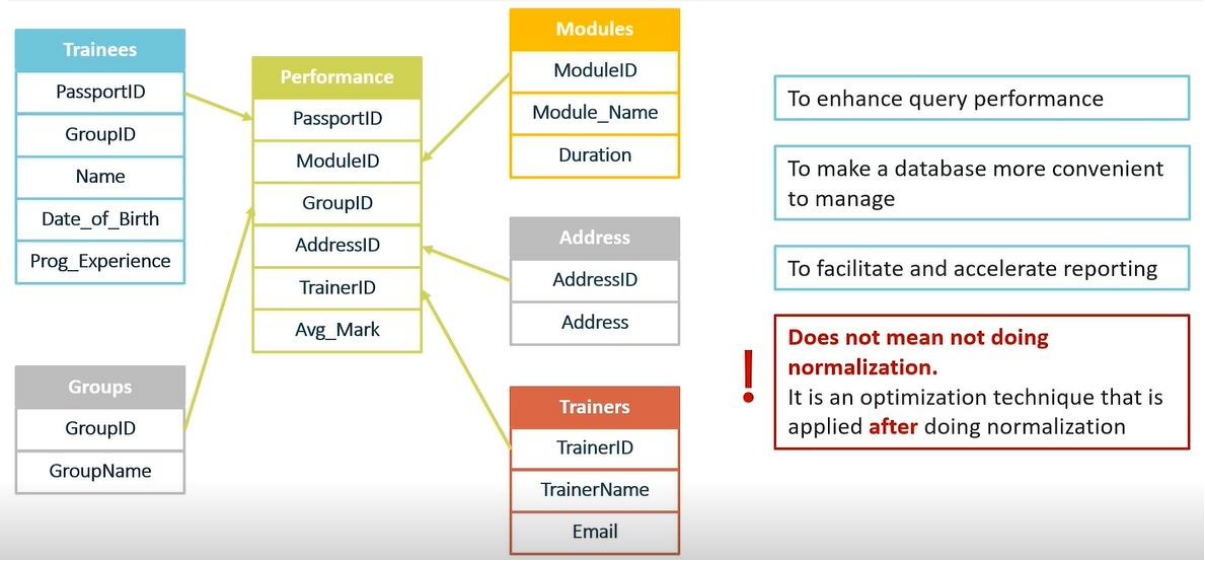
Address	AddressPhone
K1/1	+375299876543
DM3	+375293456789

Disadvantages of Normalization

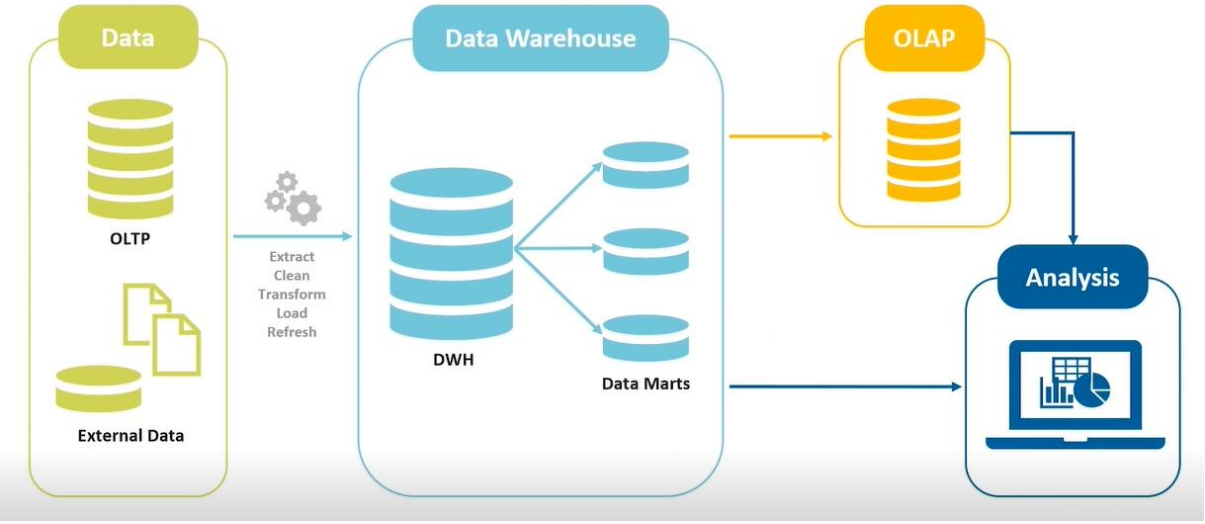


- A completely normalized database needs clear and broad understanding of the business
- More tables to join: by spreading out your data into more tables, you increase the need to join tables
- The data model is optimized for applications, not for Ad-hoc querying

Denormalization



Data Warehousing



https://www.youtube.com/playlist?list=PLSE8ODhjZXjZaHA6QcxDfJ0SIWBzQFKEG

https://www.diffen.com/difference/Data_vs_Information

https://www.lucidchart.com/pages/database-diagram/database-models?a=1

https://www.guru99.com/data-modelling-conceptual-logical.html

https://sqlrelease.com/sql-server-tutorial/types-of-keys

https://www.mssqltips.com/sqlservertip/5431/surrogate-key-vs-natural-key-differences-and-when-to-use-in-sql-server/

https://www.tutorialspoint.com/Types-of-dependencies-in-DBMS

https://www.youtube.com/watch?v=TWAiz3BWHsw

https://www.ionos.com/digitalguide/hosting/technical-matters/database-normalization/

https://rubygarage.org/blog/database-denormalization-with-examples

https://www.red-gate.com/simple-talk/databases/theory-and-design/codds-twelve-rules/

1/1 point

Keyboard Help

Relate the concepts of the theory of relational databases with objects and concepts of DataBase in practice:

cardinality	tuple	relation	attribute	degree
the number of rows in the table	row	table	column	the number of columns in the table

Pick the right statements about DBMS:

- [ ] DBMS can't be used to protect your data from accidents
- [ ] DBMSs do not isolate the operations of users simultaneously working with the database
- [x] DBMS allows control user access to the data
- [x] DBMS allows you to manipulate the data: insert, update, delete, select

✔

Predefining rules and restrictions in DataBases that are enforced in a single column or multiple columns are:

- ○ Manifestos
- ◉ Constraints
- ○ Database Manuals
- ○ Instructions

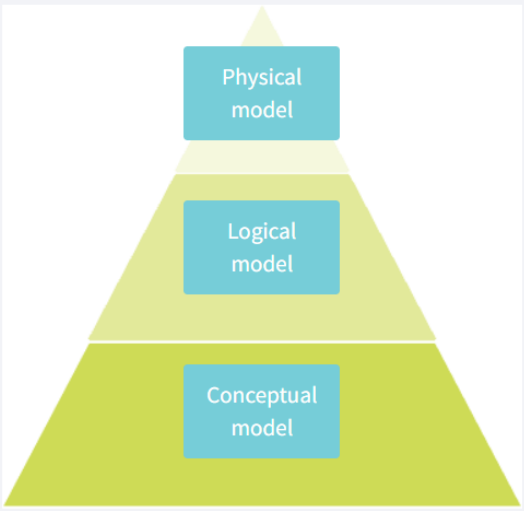
✔

Choose the wrong statement about Primary Key:

- ○ The primary key is always restricted by uniqueness
- ○ The primary key cannot contain null values
- ◉ Primary Key values may contain duplicates
- ○ All above are true statements

✔

Relate the following data model building steps according to the level of detail



For what purposes is DataBase normalization not needed?

- ( ) Minimize the redundancy
- (•) Increasing of data redundancy in a table for simplification of data analysis processes.
- ( ) Remove Insert, Update and Delete Anomaly

✔

What are the rules of first normal form?

- [x] Each table cell should contain a single (atomic) value
- [ ] Every non-key column is fully dependent on the primary key
- [x] Each row in a table needs to be unique
- [ ] There is no transitive dependency for non-prime attributes
- [x] Values stored in a column should be of the same datatype
- [x] All the columns in a table should have unique names
- [x] The order in which data is stored, does not matter
- [ ] All above are the rules of first normal form

✔

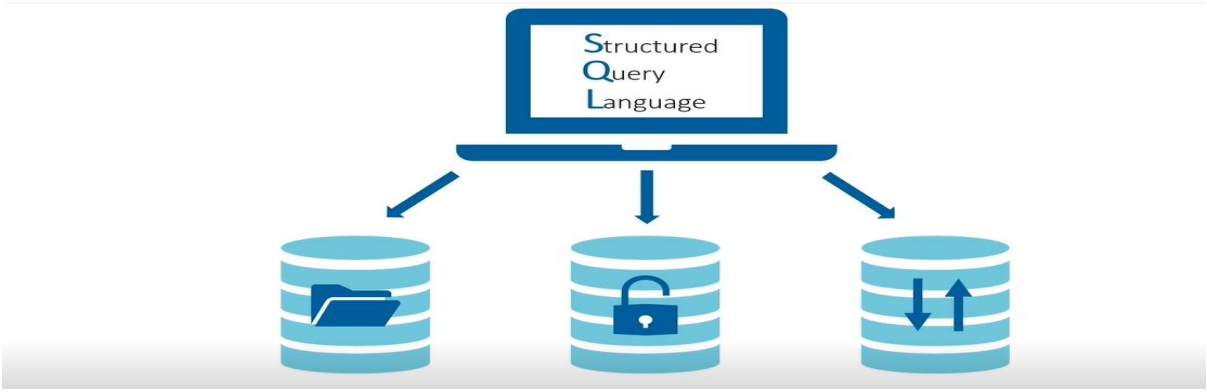
Choose the right statement(s):

- [ ] For OLAP databases, a high level of normalization is important
- [ ] If the level of normalization is higher, the less relationships appear and the more connections between them
- [x] Data warehouses are optimized for high performance of selecting a large amount of data
- [ ] For OLTP databases, the performance of inserting and updating data is not important

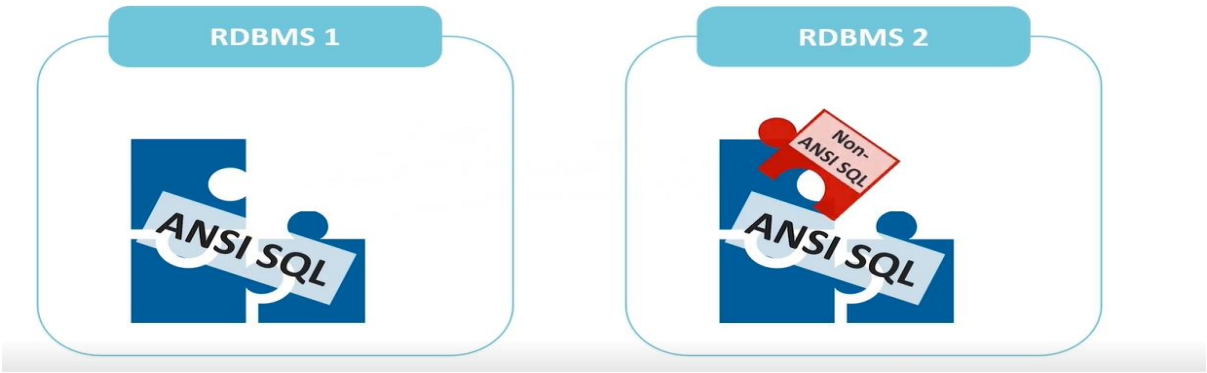
✔

- **What is SQL ?**

What is SQL?



What is SQL?



Advantages / Disadvantages of SQL

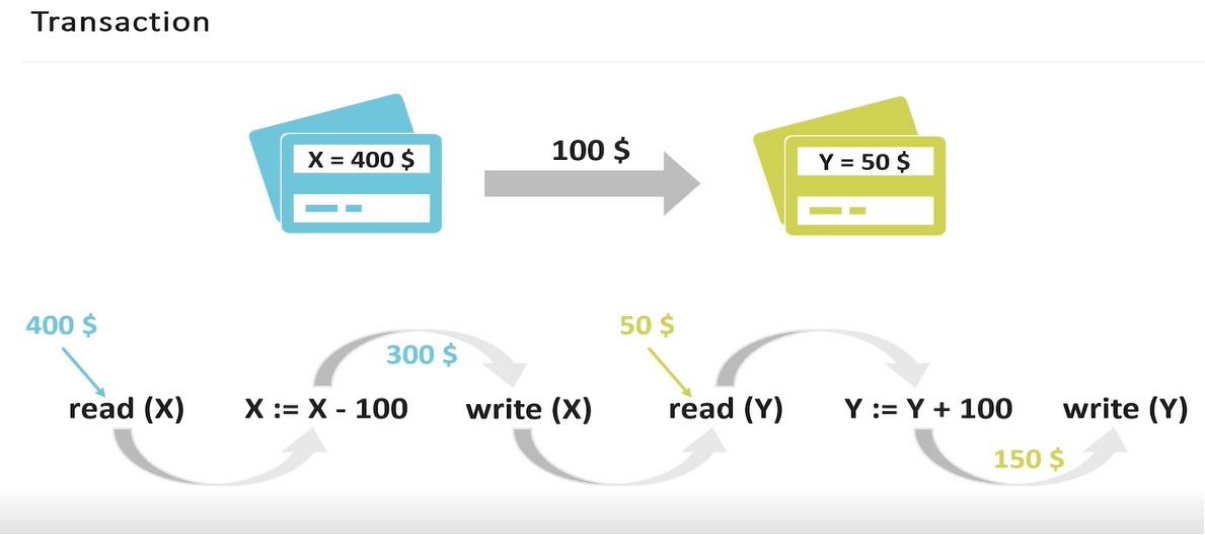
Advantages

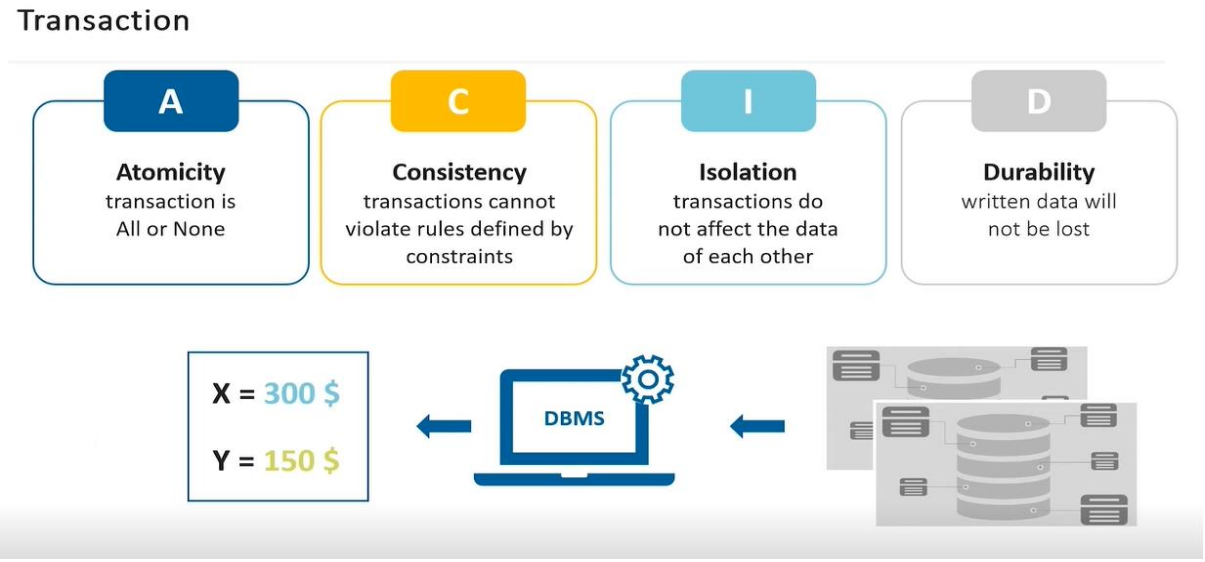
- **DBMS-independent**
- **Standardized** (ANSI SQL)
- **Declarative** (leaves query optimization to DBMS)

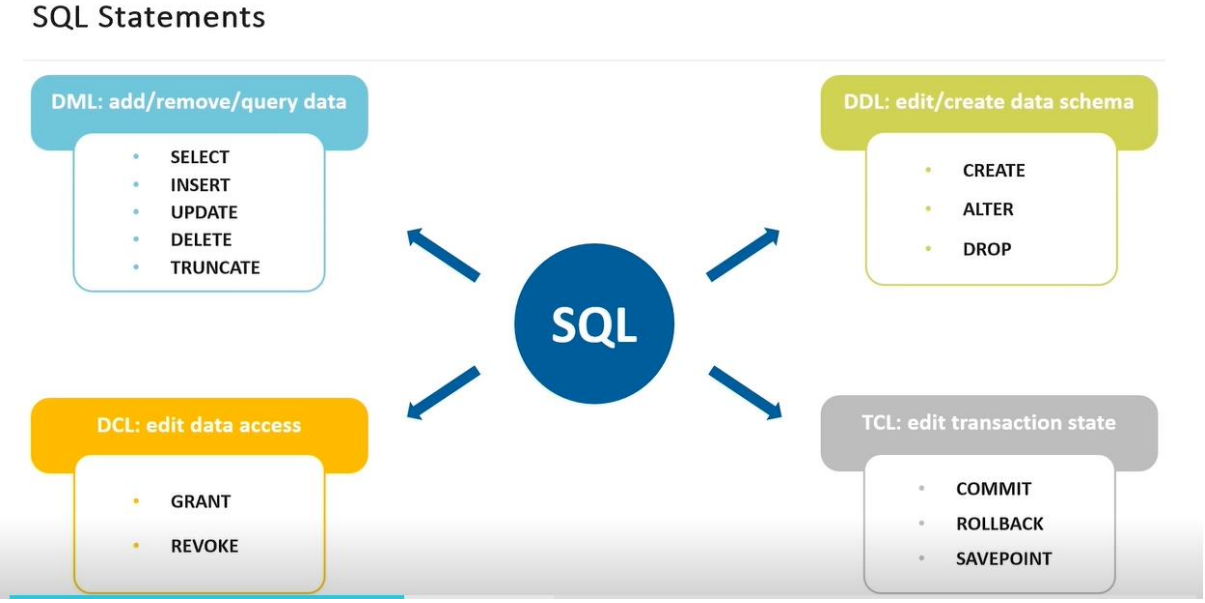
Disadvantages

- **Complexity**
- **Unpredictable performance of complex queries**
- **Requires procedural extensions** (e.g. PL/pgSQL for PostgreSQL)

- **What is transaction ?**







- **Rules of SQL**

Accepted Rules

Comments

```
SELECT *  -- '*' means 'all columns'
FROM   students;
```

The Convention

SQL keywords in UPPERCASE
for everything else - lowercase

```
SELECT *
FROM students;
```

String literals are quoted using single quotes

String literals are case-sensitive

- **Postgre SQL :**

Brief introduction to PostgreSQL

- Open source ORDBMS
- PostgreSQL is in top-5 of most popular systems in the world
- PostgreSQL has its own non-ANSI SQL commands
- **Non-ANSI SQL** command labels will be highlighted in **Green** in the course
- DBeaver, PgAdmin, SQLWorkbench/J can simplify working with PostgreSQL

Brief introduction to PostgreSQL

Features

- ACID compliance
- Support of multiple programming languages (Python, Java, C)
- Diverse customizable indexing
- Native JSON and full-text search (FTS) support
- Lots of community-developed extensions
- Advanced replication mechanisms

Datatypes :

PostgreSQL Datatypes: Numeric

Numeric DataTypes	Description	Example	Min	Max
INT	integer values	Age	-2,147,483,648	+2,147,483,647
BIGINT			-9,223,372,036,854,775,808	+9,223,372,036,854,775,807
SERIAL	automatically generate and populate values - **auto-increment**	ID (PK)	1	+2,147,483,647
BIGSERIAL			1	+9,223,372,036,854,775,807
REAL / FLOAT4	floating-point values	Surplus	1E-37	1E+37
DOUBLE PRECESION / FLOAT8			1E-307	1E+308
DECIMAL (precision, scale)	exact numeric of selectable precision	Total Weight	1234.5678 ← scale = 4 precision = 8	

PostgreSQL Datatypes: Character

Character DataTypes	Description	Example	
CHAR (n)	fixed-length, blank padded	char(1)	'Y'
VARCHAR (n)	variable-length with limit	varchar(3)	'Yes please'
TEXT	variable unlimited length	text	'Yes please ...'

PostgreSQL Datatypes: Temporal

Temporal DataTypes	Description	Example	
DATE	calendar date (year, month, day)	yyyy-mm-dd	2019-12-31
TIME (with/without time zone)	time of day	HH:MM:SS	01:02:03
TIMESTAMP (with/without time zone)	date and time	yyyy-mm-dd HH:MM:SS	2019-12-31 01:02:03

PostgreSQL Datatypes: Others

- **BOOLEAN** (TRUE / FALSE)
- **XML**, **JSON**, **JSONB** – for semi-structured data
- Many others (geometric shapes, UUID, BIT, INTERVAL, etc.)

PostgreSQL Datatypes: Implicit Conversions

BIGINT + **FLOAT4** = **FLOAT4**

DECIMAL + **FLOAT8** = **FLOAT8**

BIGINT + **TEXT** = **BIGINT**

implicit conversions may produce unexpected results

PostgreSQL Datatypes: Explicit Conversions

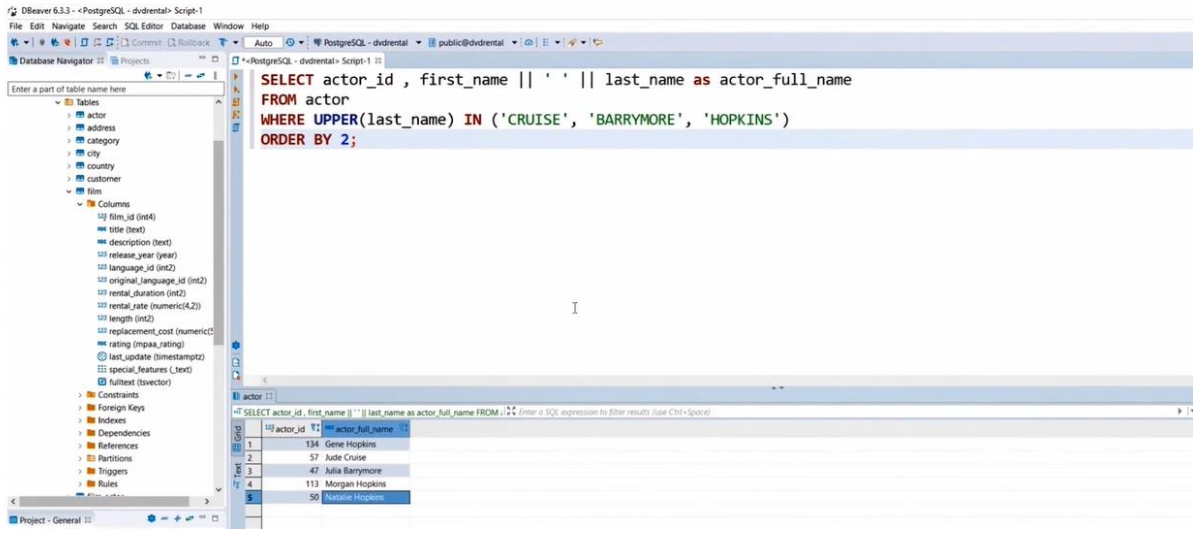
It's always better to have control over datatypes in such cases

```
SELECT  97.5 + '2.5';
```

```
SELECT  97.5 + CAST('2.5' AS DECIMAL (19,2));
```

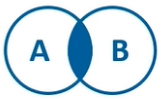
SELECT statement: Basic Elements

SELECT	DISTINCT list of columns , scalar expressions
FROM	table / joint tables / subquery
WHERE	filtering clause
GROUP BY	list of columns
HAVING	aggregation filtering clause
ORDER BY	list of columns / scalar expressions DESC
LIMIT {x}	**Returns X rows generated by the query**



Join Types: Inner Join

A	
ID	Name
1	X
2	Y
3	Z

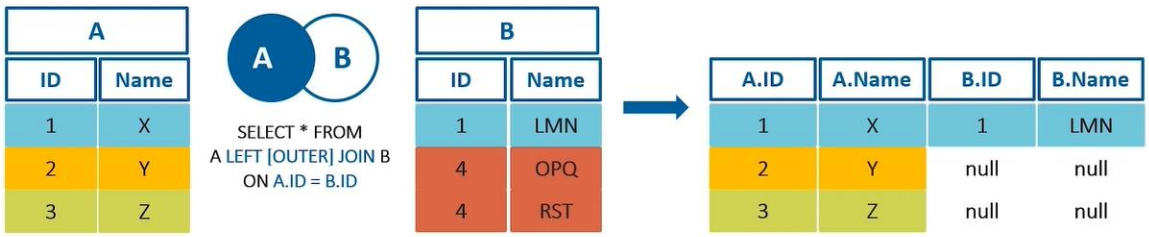


SELECT * FROM
A [INNER] JOIN B
ON A.ID = B.ID

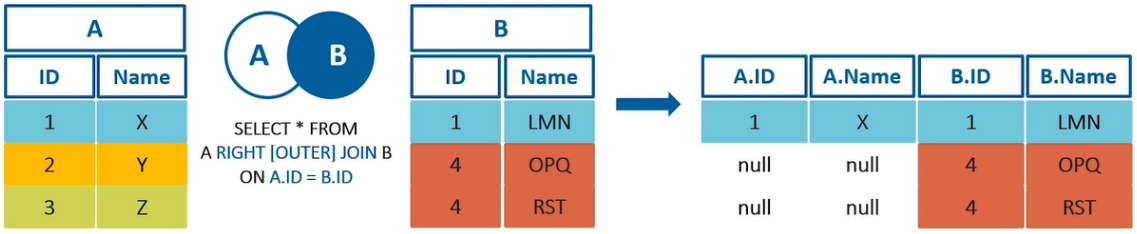
B	
ID	Name
1	LMN
4	OPQ
4	RST

A.ID	A.Name	B.ID	B.Name
1	X	1	LMN

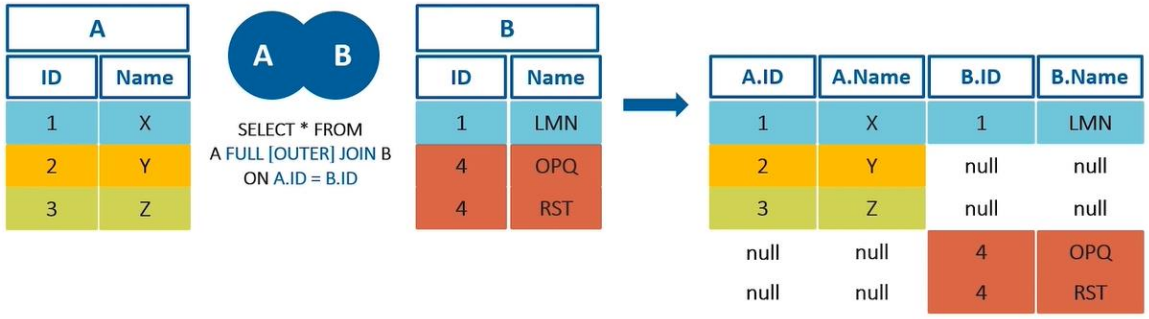
Join Types: Left Outer Join

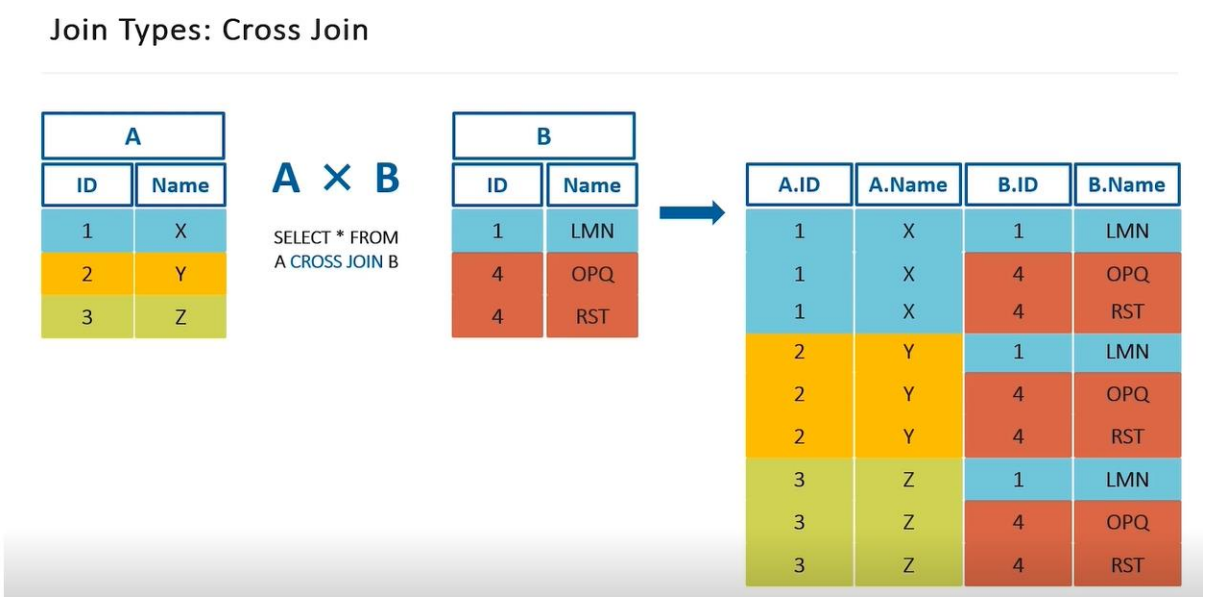


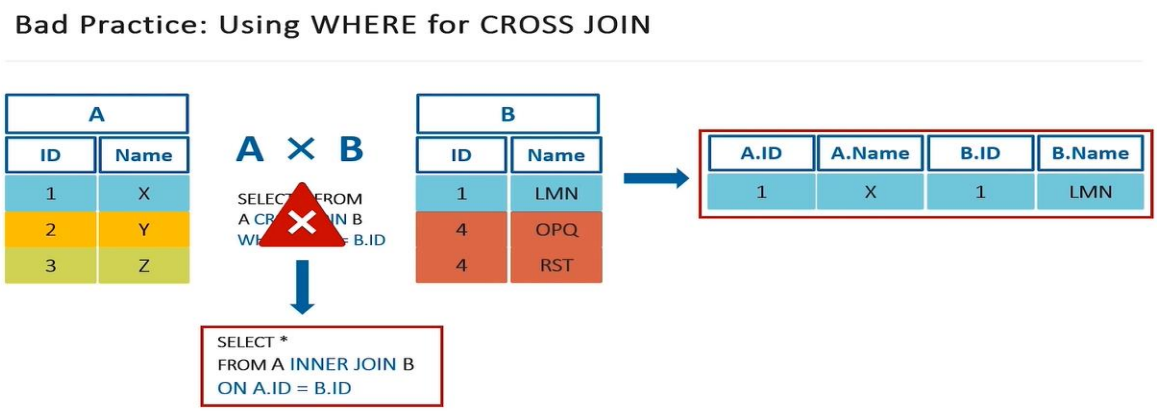
Join Types: Right Outer Join



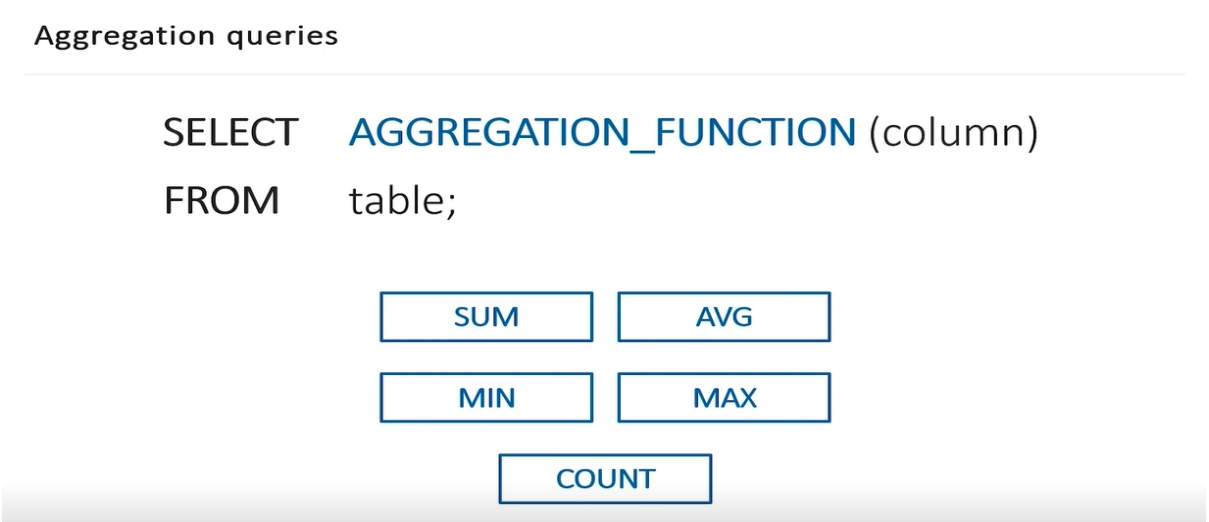
Join Types: Full Outer Join







Aggregation :



Aggregation queries

Table_A	
ID	Amount
1	null
2	null
3	null
4	null

```
SELECT  COUNT ( * )
FROM    Table_A;
```

count
4

```
SELECT  COUNT (Amount)
FROM    Table_A;
```

count
0

```
SELECT  SUM (Amount),
        AVG (Amount),
        MIN (Amount),
        MAX (Amount)
FROM    Table_A;
```

sum	avg	min	max
null	null	null	null

Aggregation

GROUP BY

```
SELECT      column_1,
            AGGREGATION_FUNCTION (column_2)
FROM        table
GROUP BY column_1;
```

GROUP BY

```
SELECT    film_id,
          rating
FROM      film
WHERE     film_id < 8
ORDER BY rating;
```

film_id	rating
4	G
2	G
5	G
1	PG
6	PG
7	PG-13
3	NC-17

```
SELECT    rating,
          COUNT(*) AS film   -- counts number of rows
FROM      film
WHERE     film_id < 8 and COUNT(*) > 1
GROUP BY rating
ORDER BY rating;
```

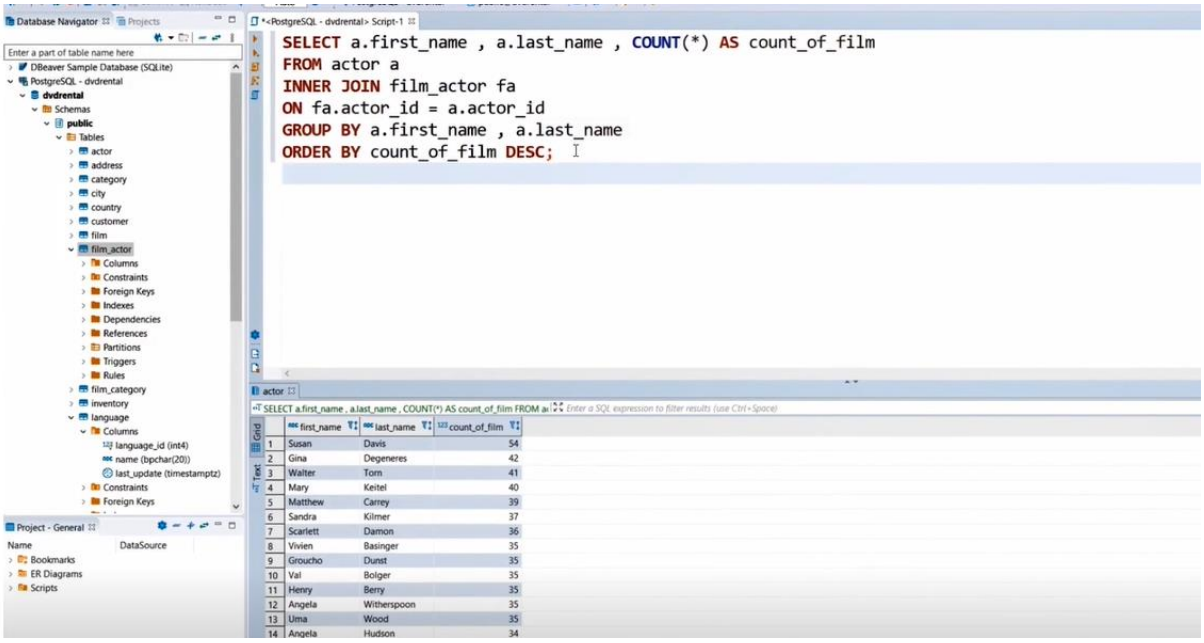
rating	films
G	3
PG	2
PG-13	1
NC-17	1

Having :

Aggregation

HAVING

```
SELECT      column_1,
            AGGREGATION_FUNCTION (column_2)
FROM        table
GROUP BY column_1
HAVING      AGGREGATION_FUNCTION (column_2) > X;
```



Subqueries :

Subqueries

```
SELECT  *
FROM
        (SELECT column_1, column_2, ...
         FROM   table_B
         ORDER BY column_1)
WHERE ...
```

Subquery

Main Query

Subqueries

```
SELECT  column_1, column_2, ...
FROM    table_A
WHERE  column_1 {OPERATOR}
                (SELECT column_3
                 FROM   table_B
                 WHERE ... );
```

=	IN
>	NOT IN
>=	ALL
<	ANY
<=	EXISTS
!=	NOT EXISTS
<>	

Subqueries

```
SELECT  *
FROM    A
WHERE  EXISTS
                (SELECT *
                 FROM   B
                 WHERE B.Name = 'LMN'
                 AND    A.ID = B.ID);
```

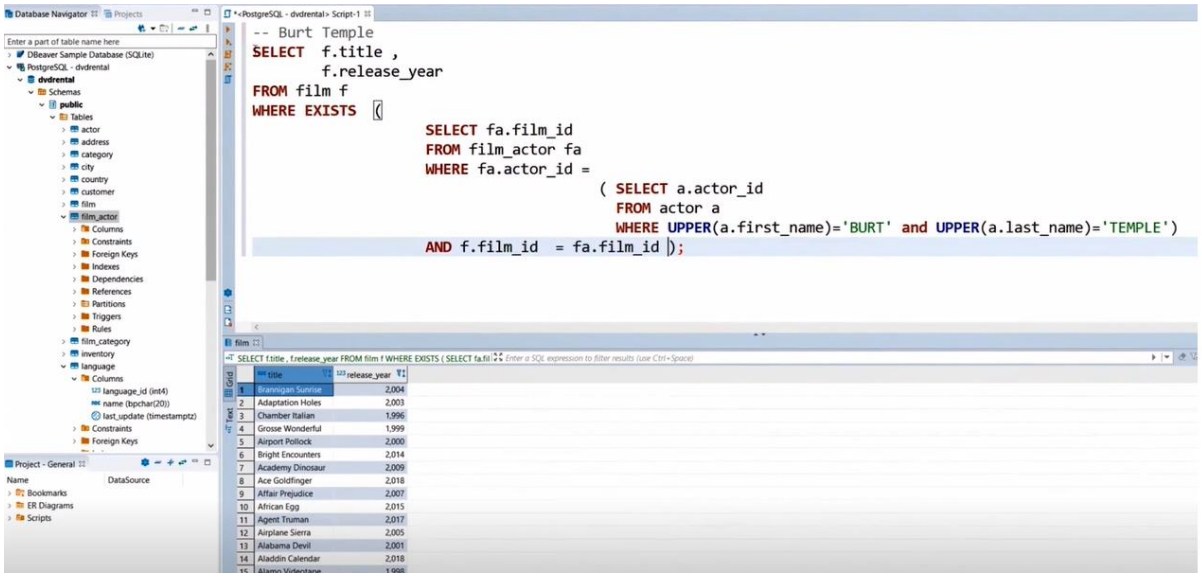
A

ID	Name
1	X
2	Y
3	Z

B

ID	Name
1	LMN
4	OPQ
2	LMN

ID	Name
1	X
2	Y



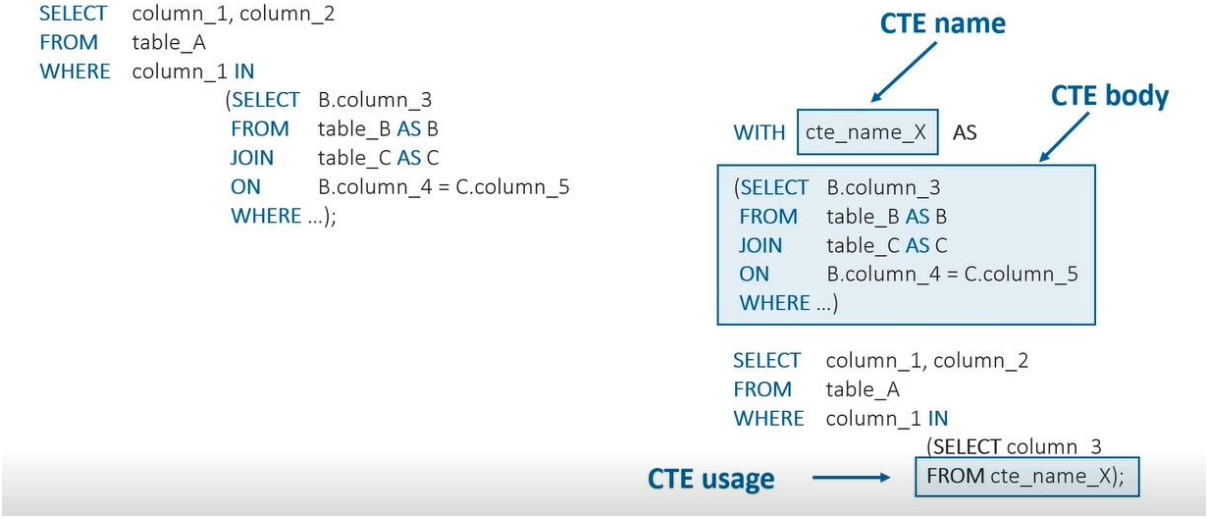
Common table Expressions :

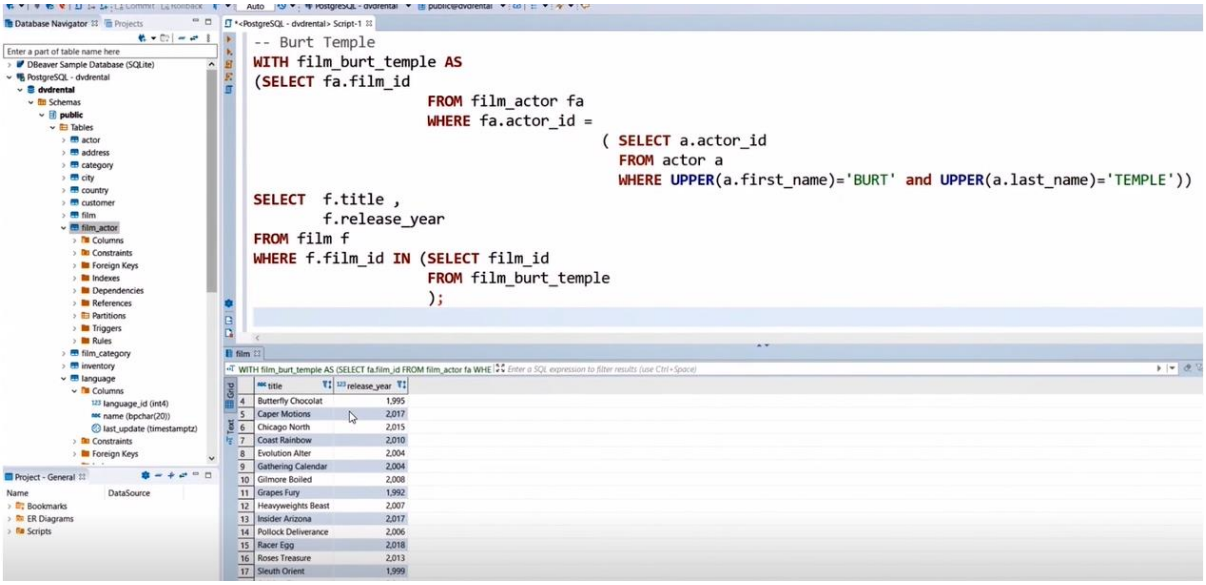
Common Table Expressions (CTE)

1	Improves readability (no need to read the query upside down)
2	Allows reusing the same subquery multiple times without duplicating the code
3	**In PostgreSQL**, CTEs can be thought of 'materialized' subqueries
4	CTEs are defined using the **WITH** clause
5	For simple single-use subqueries CTEs **are not needed**

Common Table Expressions (CTE)

```
SELECT   column_1, column_2
FROM     table_A
WHERE    column_1 IN
                (SELECT   B.column_3
                 FROM     table_B AS B
                 JOIN     table_C AS C
                 ON       B.column_4 = C.column_5
                 WHERE ...);
```





Set Operations :

Set operations. Working multiple SELECT results

Identical output structure

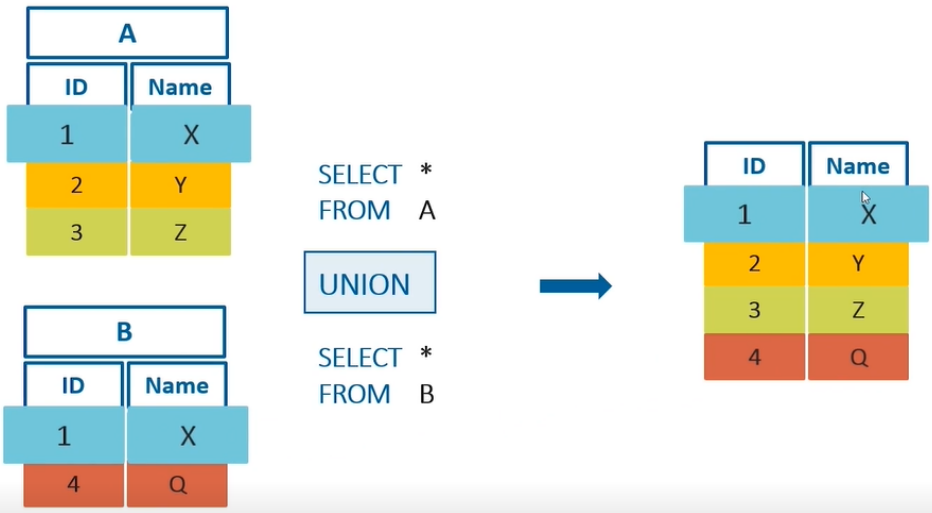
- Same number of columns
- Corresponding columns have compatible datatypes

Set Operations:

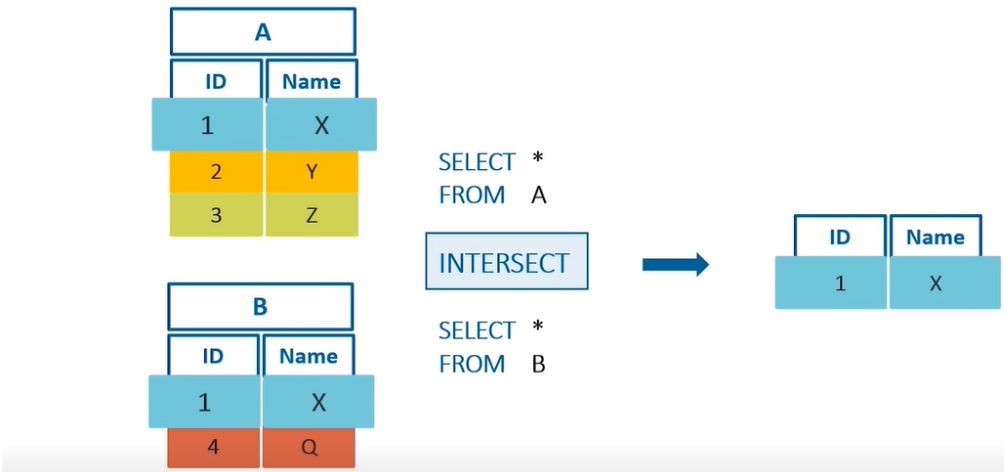
- UNION ALL
- UNION
- INTERSECT
- EXCEPT

```
SELECT column_1, column_2
FROM  table_A

{SET OPERATION_1}

SELECT column_1, column_2
FROM  table_B

{SET OPERATION_2}

SELECT column_1, column_2
FROM  table_C
```

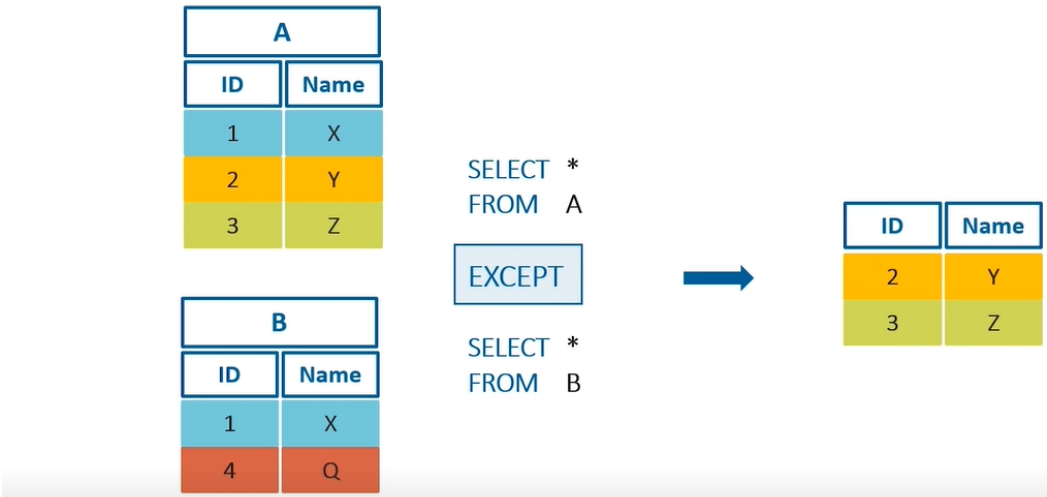
Set operations. UNION

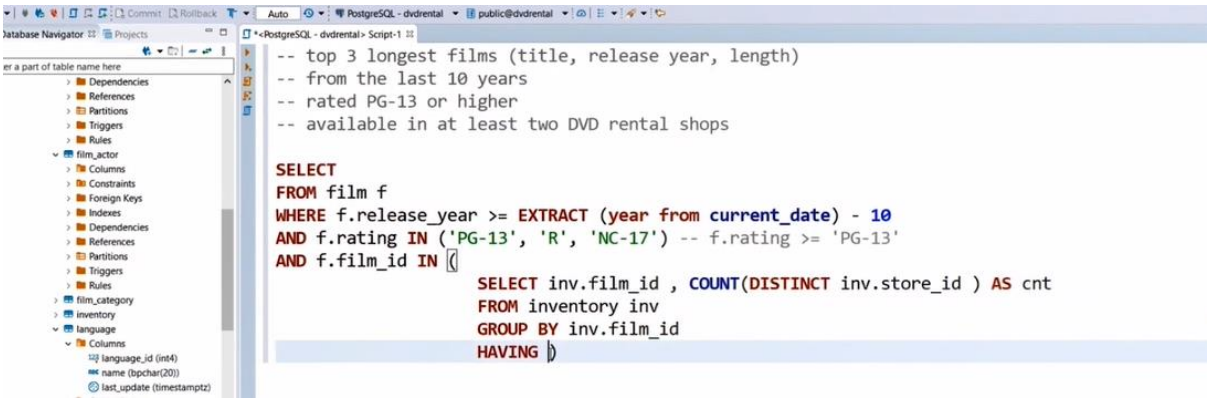


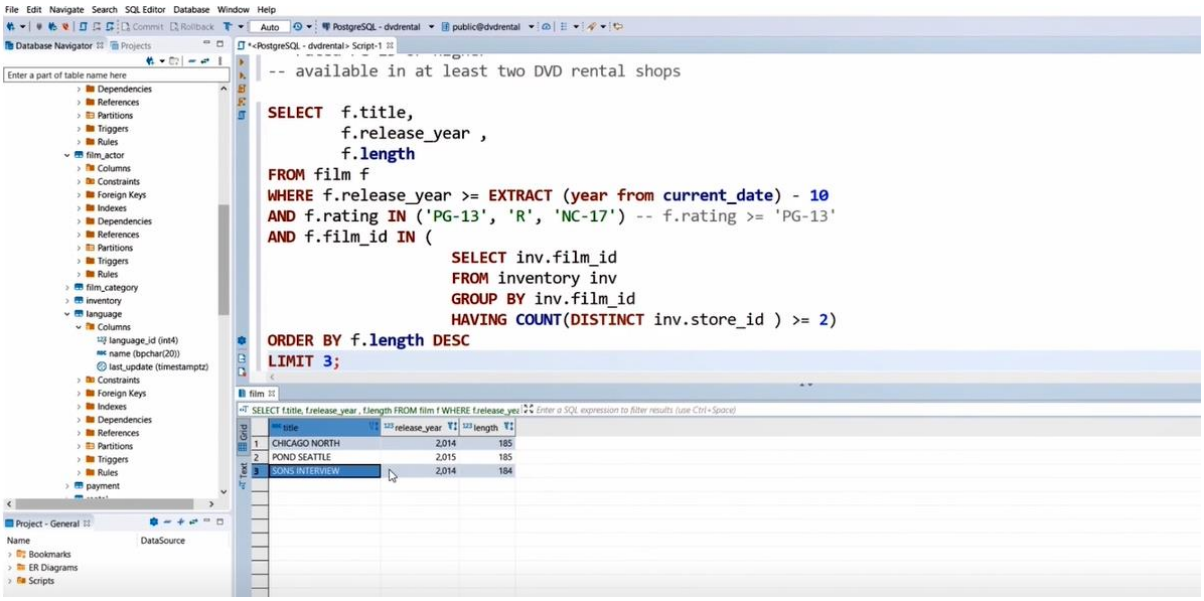
Set operations. INTERSECT



Set operations. EXCEPT







https://www.postgresql.org/docs/current/index.html

https://www.mssqltips.com/sqlservertip/6013/sql-server-in-vs-exists/

https://towardsdatascience.com/five-best-practices-for-writing-clean-and-professional-sql-code-2b081d8f7098

https://www.youtube.com/playlist?list=PLD20298E653A970F8

https://www.pgexercises.com/questions/basic/

https://sqlbolt.com/

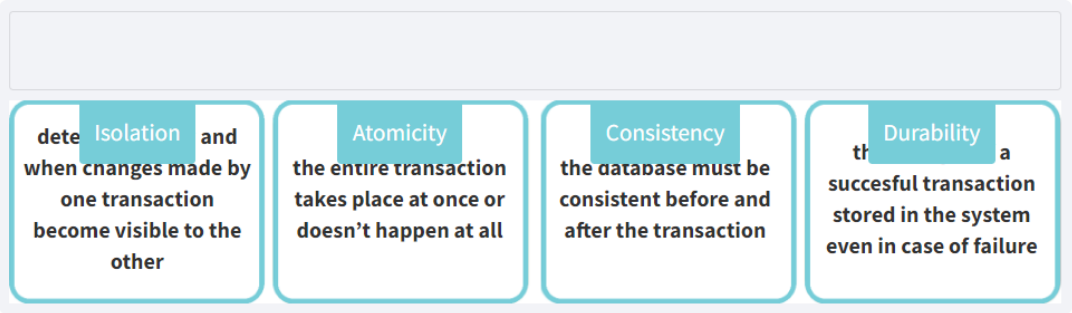
https://leetcode.com/problemset/database/

TRUNCATE command can be treated as a DML command (since it manipulates the data and can be rolled back) and a DDL command (modifies DBMS dictionary, as it can reset sequences, and it's not MVCC-safe). Some DBMS treat TRUNCATE as a DDL command (for example, Oracle or MSSQL Server). PostgreSQL does not have this clear distinction

1/1 point

Keyboard Help

Relate the definitions of each property of database transaction with the corresponding name:



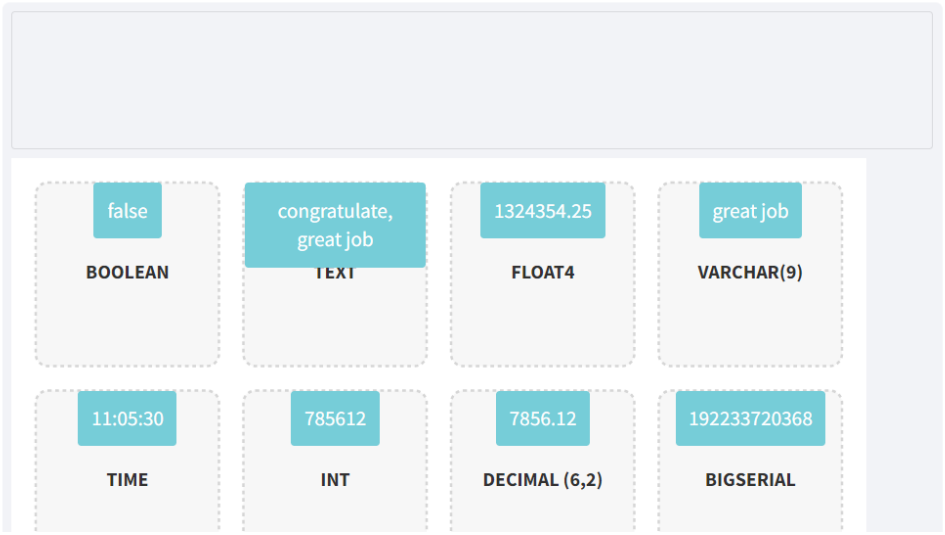
Choose the wrong statement(s):

- [ ] "--" characters in SQL means single line comment
- [ ] SQL commands can span multiple rows
- [x] SQL is case-sensitive language
- [ ] In SQL string literals are quoted using single quotes.

✔

Keyboard Help

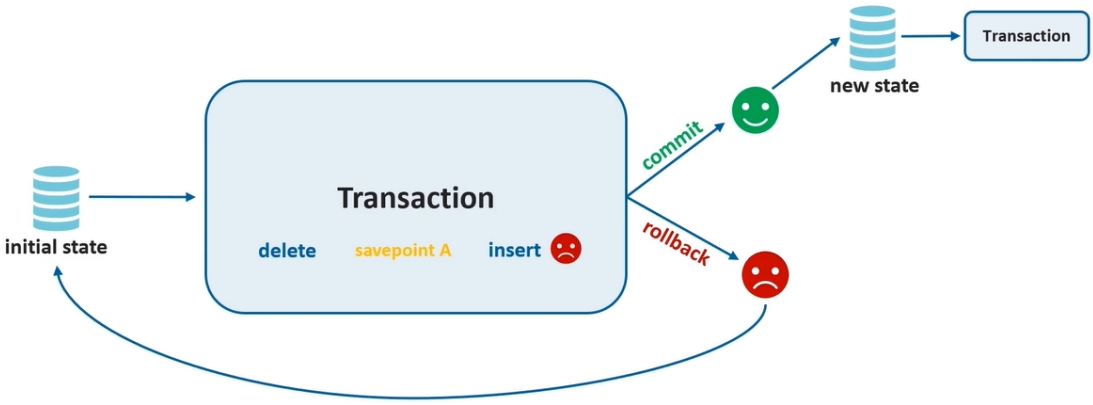
Relate examples of data with the following data types:



- **TCL Statements :**

TCL Statements

Managing transaction state



Managing transaction state. TCL statements

DML and DDL commands are transaction-safe and can be ROLLBACK'ed if necessary

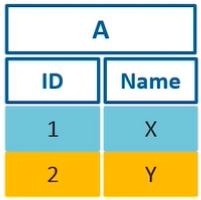
Database-wide and tablespace-wide operations (including DCL operations) **cannot** be ROLLBACK'ed

Any non-committed transaction will automatically be rolled back if session is interrupted

DML commands **except TRUNCATE** are MVCC-safe

- **DML statements :**

DML statements



Add new rows to the table (INSERT)

Update column values of particular rows in the table (UPDATE)

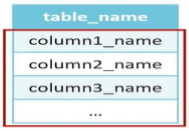
Remove particular rows from the table (DELETE or TRUNCATE)

DML command can only manipulate data, and cannot **ALTER** data schema

- **Insert :**

INSERT

```
INSERT INTO table_name
VALUES ( column1_value, column2_value, …),
       ( column1_value, column2_value, …),
... ;
```



INSERT

```
WITH inserted AS (
INSERT INTO table_name
VALUES (column1_value, column2_value, …), ...
RETURNING 1    )
SELECT COUNT (*)
FROM inserted;

INSERT INTO table_name
VALUES (column1_value, column2_value, …), ...
RETURNING ID;
```

To get the LAST id after inserting a new row

- **Update :**

```
UPDATE table_name
SET    column1_name = value1/expression1/scalar_subquery1,
       column2_name = value2/expression2/scalar_subquery2;
```

ALL the rows in the table are updated

```
UPDATE table_name
SET    column1_name = value1/expression1/scalar_subquery1,
       column2_name = value2/expression2/scalar_subquery2
WHERE  condition;
```

determine which rows you want to UPDATE

- **Delete :**

```
DELETE
```

```
DELETE FROM table_name
```

ALL the rows in the table are deleted

```
DELETE FROM table_name
WHERE  condition;
```

determine which rows you want to DELETE

- **Truncate :**

```
TRUNCATE table_name;
```

❗ Removes all rows from a table without scanning it

DELETE vs TRUNCATE

	DELETE	TRUNCATE
Effect	Removes specified rows	Removes ALL rows
Performance	Depending on data volume	Instant
Server impact	Depending on data volume	Minimal
Table Lock	Row Exclusive	Access Exclusive
Transactional	True	
MVCC-safe	True	False, effect visible to other transactions instantly

Multiple DML Statements in a query

```
WITH
     updated_rows AS (UPDATE … RETURNING 1),
     deleted_rows AS (DELETE … RETURNING 1)

SELECT    'Updated rows' AS metric_name,
          COUNT (*) AS cnt_rows
FROM      updated_rows

UNION ALL

SELECT    'Deleted rows' AS metric_name,
          COUNT (*) AS cnt_rows
FROM      deleted_rows;
```

https://www.lob.com/blog/truncate-vs-delete-efficiently-clearing-data-from-a-postgres-table

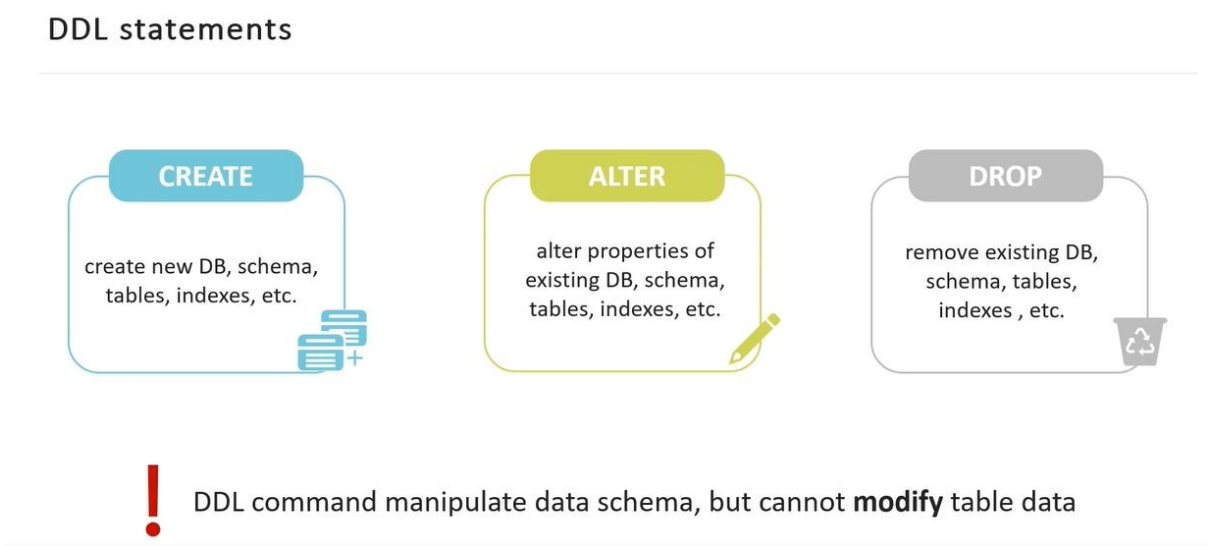
https://www.postgresql.org/docs/12/sql-vacuum.html

Choose the right statement(s) about COMMIT comand:

- [ ] defines the start of a transaction block
- [ ] creates a point within the transaction to which the transaction can be returned
- [ ] restores the database to last fixed state
- [x] saves changes which were made during transaction into the database

✔

- **DDL Statements :**



CREATE

```
CREATE DATABASE database_name [WITH options];

CREATE ROLE role_name [WITH options];

CREATE SCHEMA schema_name;

CREATE INDEX index_name
ON table_name(list_of_columns)
USING index_method;
```


```
CREATE TABLE table_name (
    column1_name DATA TYPE column1_constraint,
    column2_name DATA TYPE column2_constraint
)
```

- NOT NULL
- UNIQUE
- PRIMARY KEY
- REFERENCES

CREATE TABLE

dim_groups	
group_id	BIGSERIAL
group_number	INTEGER
creation_year	INTEGER
disband_year	INTEGER

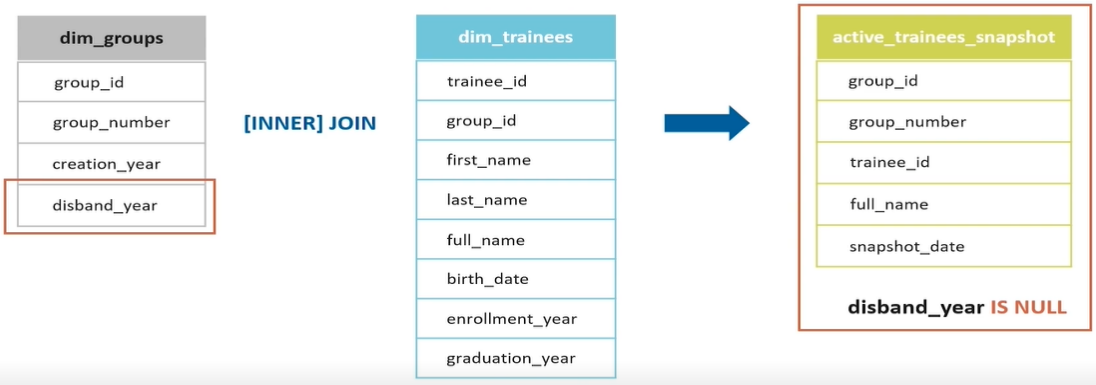
```
CREATE TABLE dim_groups
(
    group_id      BIGSERIAL PRIMARY KEY , -- constraint
    group_number  INTEGER NOT NULL, -- natural key
    creation_year INTEGER DEFAULT EXTRACT(year FROM current_date)
    NOT NULL ,
    disband_year  INTEGER
);
```

```sql
CREATE TABLE dim_trainees
(
    trainee_id  BIGSERIAL PRIMARY KEY,
    group_id    BIGINT NOT NULL REFERENCES dim_groups,
    first_name  TEXT NOT NULL,
    last_name   TEXT NOT NULL,
    full_name   TEXT GENERATED ALWAYS AS (first_name || ' ' ||
    last_name) STORED NOT NULL,
    birth_date  DATE NOT NULL,
    enrollment_year INTEGER DEFAULT EXTRACT(year FROM
    current_date) NOT NULL,
    graduation_year INTEGER
);
```

dim_trainees	
trainee_id	BIGSERIAL
group_id	BIGINT
first_name	TEXT
last_name	TEXT
full_name	TEXT
birth_date	DATE
enrollment_year	INTEGER
graduation_year	INTEGER

CREATE TABLE. Snapshot vs VIEW

CREATE TABLE



CREATE TABLE

```sql
CREATE TABLE active_trainees_snapshot AS

SELECT g.group_id,
       g.group_number,
       t.trainee_id,
       t.full_name,
       current_date AS snapshot_date
FROM   dim_groups g
JOIN   dim_trainees t
ON     g.group_id = t.group_id
WHERE  g.disband_year IS NULL;
```

active_trainees_snapshot
group_id
group_number
trainee_id
full_name
snapshot_date

View vs table :

CREATE VIEW

VIEW

```
CREATE VIEW active_trainees_snapshot AS

SELECT  g.group_id,
        g.group_number,
        t.trainee_id,
        t.full_name,
        current_date AS snapshot_date
FROM    dim_groups g
JOIN    dim_trainees t
ON      g.group_id = t.group_id
WHERE   g.disband_year IS NULL;
```

TABLE

```
CREATE TABLE active_trainees_snapshot AS

SELECT  g.group_id,
        g.group_number,
        t.trainee_id,
        t.full_name,
        current_date AS snapshot_date
FROM    dim_groups g
JOIN    dim_trainees t
USING   g.group_id = t.group_id
WHERE   g.disband_year IS NULL;
```

CREATE VIEW: VIEW vs TABLE

VIEW	TABLE
Only hold the query which references certain data projection	Hold the data
Changes (altering/dropping/...) does not affect the data in the database	Changes (altering/dropping/...) can affect existing data

Alter command :

ALTER: Add a column

```
ALTER TABLE table_name ADD COLUMN new_column_name DATA TYPE;
```

dim_trainees	
trainee_id	BIGSERIAL
group_id	BIGINT
first_name	TEXT
last_name	TEXT
birth_date	DATE
graduation_year	INTEGER
education	TEXT

```
ALTER TABLE   dim_trainees
ADD COLUMN    education TEXT;
```

ALTER: Remove a column

```
ALTER TABLE table_name DROP COLUMN column_name;
```

dim_trainees	
trainee_id	BIGSERIAL
group_id	BIGINT
first_name	TEXT
last_name	TEXT
birth_date	DATE
graduation_year	INTEGER
education	TEXT

```
ALTER TABLE   dim_trainees
DROP COLUMN   birth_date;
```

ALTER: Rename a column

```
ALTER TABLE table_name RENAME COLUMN column_name TO new_column_name;
```

dim_trainees	
trainee_id	BIGSERIAL
group_id	BIGINT
first_name	TEXT
last_name	TEXT
graduation_date	INTEGER
education	TEXT

```
ALTER TABLE   dim_trainees
RENAME COLUMN graduation_year TO graduation_date;
```

ALTER: Change the DATA TYPE of the column

```
ALTER TABLE table_name ALTER COLUMN column_name TYPE {DATA TYPE} [USING clause];
```

dim_trainees	
trainee_id	BIGSERIAL
group_id	BIGINT
first_name	TEXT
last_name	TEXT
graduation_date	DATE
education	TEXT

```
ALTER TABLE   dim_trainees
ALTER COLUMN  graduation_date TYPE DATE
USING (graduation_date || '-01-01')::DATE ;
```

ALTER: Change a default value for the column

```
ALTER TABLE table_name ALTER COLUMN column_name SET DEFAULT value | DROP DEFAULT;
```

dim_trainees	
trainee_id	BIGSERIAL
group_id	BIGINT
first_name	TEXT
last_name	TEXT
graduation_date	DATE
education	TEXT

```
ALTER TABLE   dim_trainees
ALTER COLUMN  graduation_date
SET DEFAULT   '1900-01-01';
```

```
INSERT INTO dim_trainees
VALUES(3, 'Ivan');
```

trainee_id	group_id	first_name	last_name	graduation_date	education
1	3	Ivan	null	1900-01-01	null

ALTER: Manage constraints

```
ALTER TABLE table_name ALTER COLUMN column_name SET NOT NULL | DROP NOT NULL;
```

dim_trainees	
trainee_id	BIGSERIAL
group_id	BIGINT
first_name	TEXT
last_name	TEXT
graduation_date	DATE
education	TEXT

```
ALTER TABLE   dim_trainees
ALTER COLUMN  last_name SET NOT NULL;
```

```
INSERT INTO dim_trainees(group_id, first_name, last_name)
VALUES(3, 'Ivan', 'Ivanov');
```

trainee_id	group_id	first_name	last_name	graduation_date	education
1	3	Ivan	Ivanov	1900-01-01	null

ALTER: Manage constraints

```
ALTER TABLE table_name ADD CHECK constraint_definition;
```

dim_trainees	
trainee_id	BIGSERIAL
group_id	BIGINT
first_name	TEXT
last_name	TEXT
graduation_date	DATE
education	TEXT

```
ALTER TABLE   dim_trainees
ADD CHECK     (education in ('completed', 'incomplete'));
```

```
INSERT INTO dim_trainees (group_id, first_name, last_name,
                          graduation_date, education)
VALUES(3, 'Ivan', 'Ivanov', '2019-07-07', 'completed');
```

trainee_id	group_id	first_name	last_name	graduation_date	education
1	3	Ivan	null	2019-07-07	completed

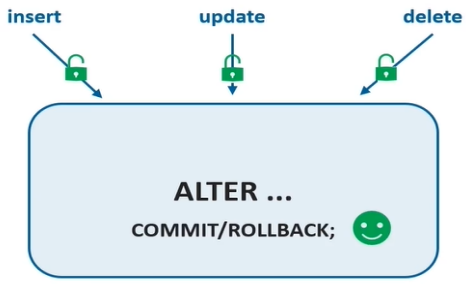
ALTER: Rename DB objects

```
ALTER TABLE table_name RENAME TO new_table_name;
```

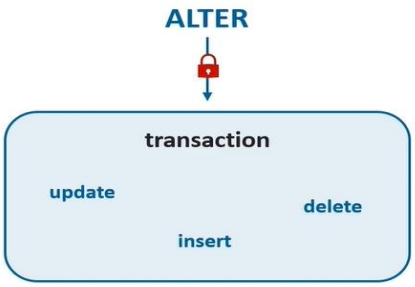
dim_participants	
trainee_id	BIGSERIAL
group_id	BIGINT
first_name	TEXT
last_name	TEXT
graduation_date	DATE
education	TEXT

```
ALTER TABLE    dim_trainees
RENAME TO      dim_participants;
```

ALTER



ALTER



ALTER

Drop command :

DROP - Be careful!!!

DROP TABLE removes the table and underlying data

DROP is a very dangerous command and should almost **never** be used in Production environment, especially on business-critical tables.

Only superuser, schema owner, and table owner have privilege to remove the table

It is better to rename a table and move it to archive schema and/or data storage than to DROP it.

DROP

```
DROP TABLE [IF EXISTS] table_name [CASCADE];
```

```
DROP TABLE    dim_modules;
```
ERROR: table doesn't exist

```
DROP TABLE IF EXISTS dim_trainees;
```

DROP

```
DROP TABLE [IF EXISTS] table_name [CASCADE | RESTRICT];
```

```
DROP TABLE dim_groups;
```
ERROR: cannot drop table author because other objects depend on it

dim_groups	
group_id	BIGSERIAL
group_number	INTEGER
creation_year	INTEGER
disband_year	INTEGER

dim_trainees	
trainee_id	BIGSERIAL
group_id	BIGINT
first_name	TEXT
last_name	TEXT
graduation_date	DATE
education	TEXT

DROP

```
DROP TABLE [IF EXISTS] table_name [CASCADE | RESTRICT];
```

```
DROP TABLE dim_groups CASCADE;
```

Choose the right statement(s) about DDL:

- [ ] DDL commands manipulate data schema and modify table data
- [x] DDL commands manipulate data schema, but cannot modify table data
- [ ] DDL commands modify table data, but cannot manipulate data schema
- [ ] DDL commands are: UPDATE, CREATE, DROP, ALTER

✔

What are the distinctive features of view from table?

- [x] view retrieves data at the time of access instead of storing it on disk
- [ ] view holds the data
- [x] executing DDL commands on views does not affect data in the database
- [x] view holds the query which references certain data projection
- [ ] executing DDL commands on views affect existing data

✔

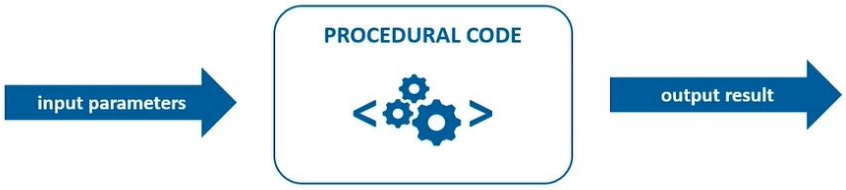
Choose the correct syntax for Customer table:

- [ ] ADD COLUMN Customer.CustomerName TEXT;
- [x] ALTER TABLE Customer ADD CHECK (CustomerNumber>0);
- [ ] DROP COLUMN Customer.CustomerName;
- [ ] ALTER TABLE Customer ALTER COLUMN CustomerNumber ADD CHECK (CustomerNumber>0);
- [x] ALTER TABLE Customer ADD COLUMN CustomerName TEXT;
- [ ] ALTER COLUMN Customer.CustomerName SET DEFAULT 'N/A';
- [x] ALTER TABLE Customer DROP COLUMN CustomerName;
- [x] ALTER TABLE Customer ALTER COLUMN CustomerName SET DEFAULT 'N/A';
- [x] DROP TABLE Customer;

✔

- **DDL : Create function :**

CREATE FUNCTION



CREATE FUNCTION

```
CREATE [OR REPLACE] FUNCTION funct_name ([IN/OUT/INOUT] arg1_name DATA TYPE,
                                         [IN/OUT/INOUT] arg2_name DATA TYPE)
```

overrides existing function with the same name and order of types of function arguments

CREATE FUNCTION

```
CREATE [OR REPLACE] FUNCTION funct_name ([IN/OUT/INOUT] arg1_name DATA TYPE,
                                         [IN/OUT/INOUT] arg2_name DATA TYPE)
```

IN - the arguments are passed in and not returned (if omitted - the default is IN)
OUT - return output from a function

CREATE FUNCTION

```
CREATE [OR REPLACE] FUNCTION funct_name ([IN/OUT/INOUT] arg1_name DATA TYPE,
                                         [IN/OUT/INOUT] arg2_name DATA TYPE)
RETURNS
```

specifies the datatype of function result

CREATE FUNCTION

```
CREATE [OR REPLACE] FUNCTION funct_name ([IN/OUT/INOUT] arg1_name DATA TYPE,
                                         [IN/OUT/INOUT] arg2_name DATA TYPE)
RETURNS [SETOF] DATA TYPE / TABLE (column1_name DATA TYPE)
```

specify SETOF or TABLE if function is expected to return multiple items/rows

CREATE FUNCTION

```
CREATE [OR REPLACE] FUNCTION funct_name ([IN/OUT/INOUT] arg1_name DATA TYPE,
                                         [IN/OUT/INOUT] arg2_name DATA TYPE)
RETURNS [SETOF] DATA TYPE / TABLE (column1_name DATA TYPE)
AS $$ function_code $$
LANGUAGE lang_name
[IMMUTABLE / STABLE / VOLATILE]
```

defines whether function call can change database state

CREATE FUNCTION

```
CREATE [OR REPLACE] FUNCTION funct_name ([IN/OUT/INOUT] arg1_name DATA TYPE,
                                         [IN/OUT/INOUT] arg2_name DATA TYPE)
RETURNS [SETOF] DATA TYPE / TABLE (column1_name DATA TYPE)
AS $$ function_code $$
LANGUAGE lang_name
[IMMUTABLE / STABLE / VOLATILE]
[SECURITY INVOKER / DEFINER]
```

defines which role's privileges are used during the call

CREATE FUNCTION

```
CREATE [OR REPLACE] FUNCTION funct_name ([IN/OUT/INOUT] arg1_name DATA TYPE,
                                         [IN/OUT/INOUT] arg2_name DATA TYPE)
RETURNS [SETOF] DATA TYPE / TABLE (column1_name DATA TYPE)
AS $$ function_code $$
LANGUAGE lang_name
[IMMUTABLE / STABLE / VOLATILE]
[SECURITY INVOKER / DEFINER]
[PARALLEL SAFE / UNSAFE / RESTRICTED]
```

defines whether function can be used in parallel queries

- **CREATE FUNCTION. SQL vs PLpgsql :**

CREATE FUNCTION: Procedural Languages

SQL

parameterized views

```
SELECT   column1_name
FROM     table1_name
WHERE    condition
...
```

PL/pgSQL

block-structured language

```
[label]
[DECLARE
    declarations]
BEGIN
    statements;
END [label];
```

CREATE FUNCTION: Language SQL vs PL/pgSQL

	SQL	PL/pgSQL
Purpose	Operations on data, schema and all database objects	Construct, parameterize and orchestrate SQL statements; create reusable code for multiple SQL statements
Execution model	Declarative, parsed and executed as a whole	Imperative, executed sequentially line-by-line
Control by programmer	Limited control, database engine decides how to perform the query	Near absolute control, as the programmer writes the code to be executed
Monitoring	No intermediate states, SQL statements only provide their results, limited monitoring through database dictionary	State can be captured at any point during execution via variables and log output
Stored as DB objects	Views and within orchestrating procedures, functions and triggers	Procedures, functions, triggers
Optimized by PostgreSQL	Cost optimization, huge performance improvement	Performance mostly depends on the programmer, limited optimization
Capabilities	High, covers 90% of backend tasks	Very high, covers 99% of backend tasks with the help of SQL

- **Create Function : SQL**

CREATE FUNCTION: Language SQL

Returns only students of specified group

```sql
SELECT g.group_id,
       g.group_number,
       t.trainee_id,
       t.full_name,
       current_date AS snapshot_date
FROM   dim_groups g
JOIN   dim_trainees t
ON     g.group_id = t.group_id
WHERE  g.group_id = 10;
```

Change input parameter every time?

CREATE FUNCTION: Language SQL

```sql
CREATE FUNCTION get_trainee_group_members(IN i_group_id BIGINT DEFAULT NULL)
RETURNS TABLE (group_id BIGINT,
               group_number INTEGER,
               trainee_id BIGINT,
               full_name TEXT,
               snapshot_date DATE)
AS $$
SELECT g.group_id,
       g.group_number,
       t.trainee_id,
       t.full_name,
       current_date AS snapshot_date
FROM   dim_groups g
JOIN   dim_trainees t
ON     g.group_id = t.group_id
WHERE  g.group_id = i_group_id
OR     i_group_id IS NULL AND g.disband_year IS NULL
$$
LANGUAGE sql;
```

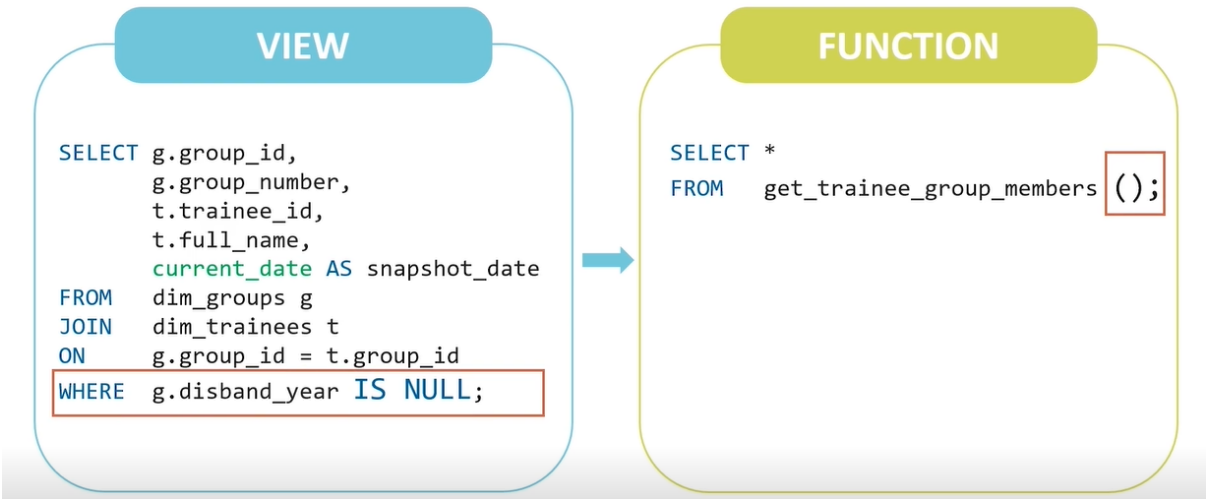
Returns only students of specified group **i_group_id**

CREATE FUNCTION: Language SQL

```sql
CREATE FUNCTION get_trainee_group_members(IN i_group_id BIGINT DEFAULT NULL)
RETURNS TABLE (group_id BIGINT,
               group_number INTEGER,
               trainee_id BIGINT,
               full_name TEXT,
               snapshot_date DATE)
AS $$
SELECT g.group_id,
       g.group_number,
       t.trainee_id,
       t.full_name,
       current_date AS snapshot_date
FROM   dim_groups g
JOIN   dim_trainees t
ON     g.group_id = t.group_id
WHERE  g.group_id = i_group_id
OR     i_group_id IS NULL AND g.disband_year IS NULL
$$
LANGUAGE sql;
```

Returns only students of specified group **i_group_id** and if the group is not specified, returns info about all active groups

CREATE FUNCTION: Language SQL



- **Create Function : PLPgSQL**

CREATE FUNCTION: Language PL/pgSQL

```
DO $$
<block_name>
```

anonymous block's name (optional)

CREATE FUNCTION: Language PL/pgSQL

```
DO $$
<block_name>  -- anonymous block's name (optional)

DECLARE
      variable1_name DATA TYPE := expression;
      variable2_name DATA TYPE := expression;
```

DECLARE block is used to declare variables to be used in the program

CREATE FUNCTION: Language PL/pgSQL

```
DO $$
<block_name>  -- anonymous block's name (optional)

DECLARE
      variable1_name DATA TYPE := expression;
      variable2_name DATA TYPE := expression;

BEGIN
      statements;
      RAISE NOTICE/EXCEPTION/WARNING 'text', variable1_name;
```

statement to report messages and raise errors

CREATE FUNCTION: Language PL/pgSQL

```
DO $$
<block_name>  -- anonymous block's name (optional)

DECLARE
      variable1_name DATA TYPE := expression;
      variable2_name DATA TYPE := expression;

BEGIN
      statements;
      RAISE NOTICE/EXCEPTION/WARNING 'text', variable1_name;

END   <block_name>; $$;
```

CREATE FUNCTION: Language PL/pgSQL

```
CREATE FUNCTION add_new_group (IN i_group_number INT)
RETURNS BIGINT
LANGUAGE plpgsql

AS $$
DECLARE
      v_group_id BIGINT;

BEGIN
      INSERT INTO dim_groups(group_number)
      VALUES (i_group_number)
      RETURNING group_id INTO v_group_id;

      RAISE NOTICE 'New group % added! ID = %',i_group_number, v_group_id;

      RETURN v_group_id;
END;
$$;
```

Choose statement(s) that refers only to the view:

- [ ] can contain several statements, loops
- [ ] parsed and executed as a whole
- [ ] executed sequentially line-by-line
- [ ] can perform modifications to one or several tables
- [x] doesn't accept parameters
- [ ] all of the above refers to functions

✔

Correct answer

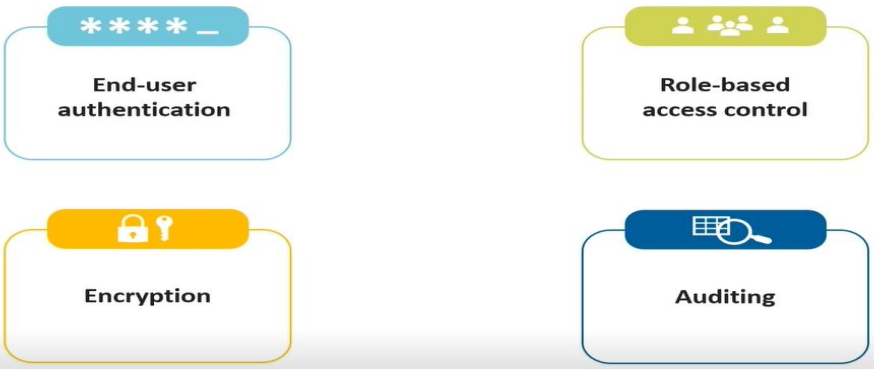
- (●)
```
DO $$
DECLARE
     var_name DATA TYPE := expression;
BEGIN
     statements;
     RAISE NOTICE/EXCEPTION 'text', var_name;
END;
$$;
```

- ( )
```
DO $$
DECLARE
     var_name := expression;
BEGIN
     statements;
     RAISE NOTICE/EXCEPTION 'text', var_name;
END;
$$;
```

✔

- **DCL Statements :**

DBMS security overview: Mechanisms



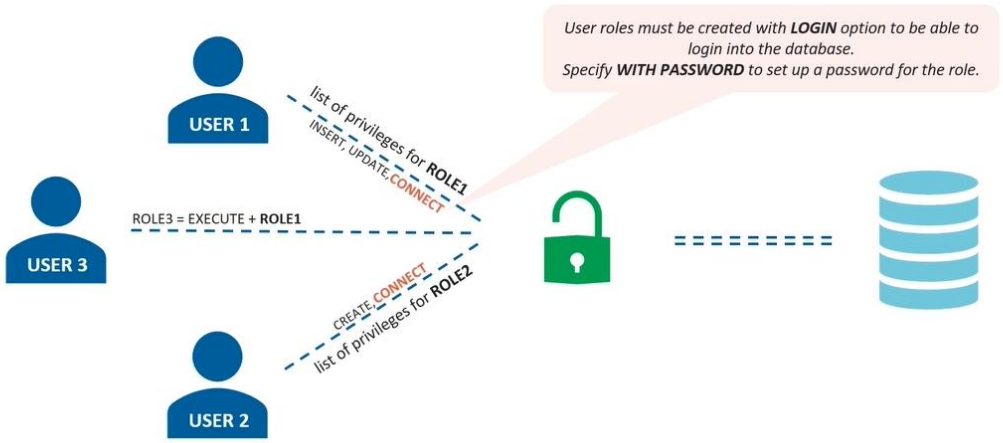
DBMS security overview: Strategies

- Careful backup management
- Ensure a strong password standard for your enterprise
- Ensure consistency checks for your data to prohibit malicious records from being inserted
- Only provide users with privileges they need (create roles with narrow scope)

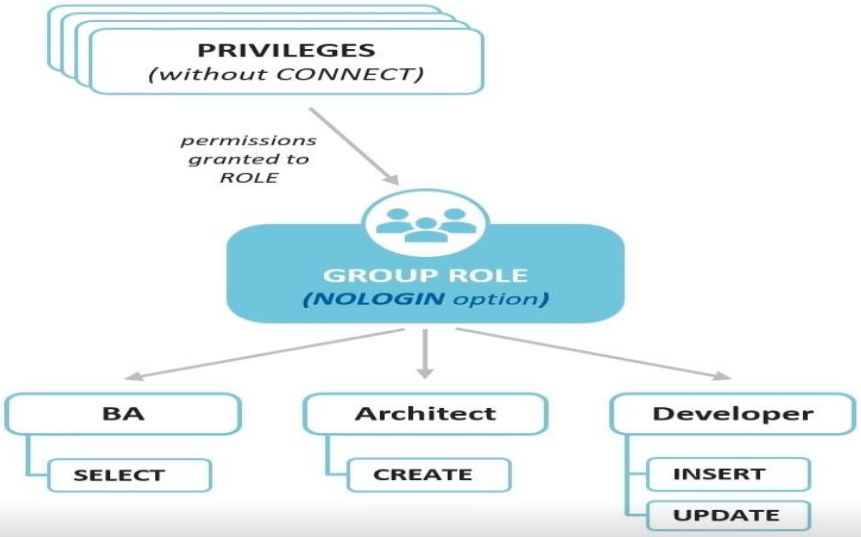
Types of privileges

Name	Purpose
SELECT	Allows retrieving data from table-like objects (tables, views, etc.)
INSERT	Allows INSERTing data into table-like objects
UPDATE	Allows UPDATEing data of table-like objects
DELETE	Allows DELETEing data of table-like objects
TRUNCATE	Allows TRUNCATEing table-like objects
CREATE	Allows CREATEing objects within specified object
REFERENCES	Allows creation of foreign key referencing specific table columns
TRIGGER	Allows creation of a trigger on a table, view, etc.
TEMPORARY	Allows temporary tables to be created while using the database
EXECUTE	Allows calling a function or procedure
USAGE	Allows usage of particular language or looking up objects on particular schema
CONNECT	Allows the grantee to connect to the database

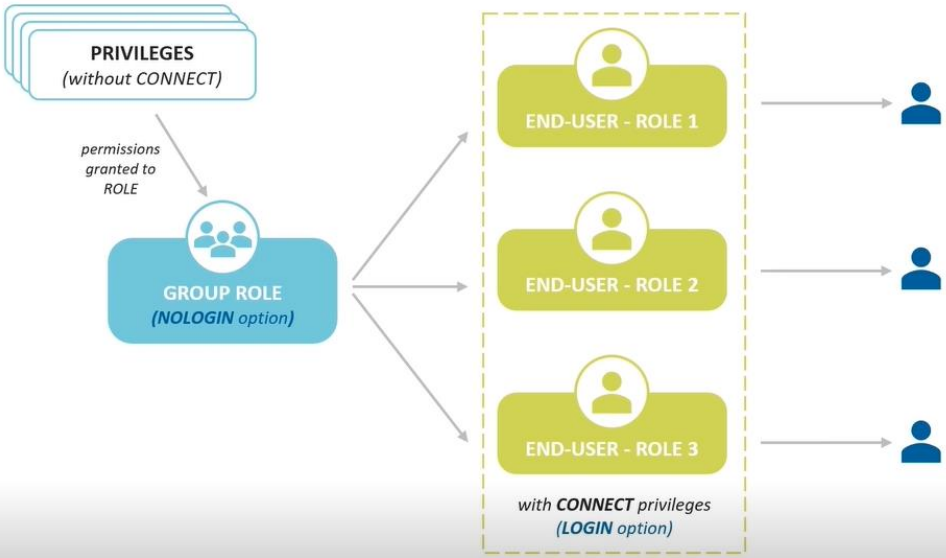
Roles



Roles



Roles



Roles

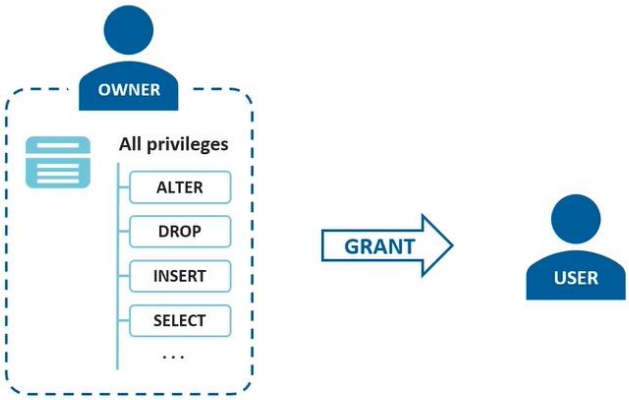
GRANT

```
GRANT   privilege
ON      object
TO      {PUBLIC|GROUP group|username}
```

REVOKE

```
REVOKE  privilege
ON      object
FROM    {PUBLIC|groupname|username}
```

Roles



! Cannot grant the permission to ALTER or DROP the database object.
That only be transferred via ALTER ... SET OWNER TO ... command.

Eg

Roles

```
GRANT SELECT ON TABLE film TO dvd_rental_client;
```

grants privilege to extract data from table film

Roles

```
GRANT SELECT ON TABLE film TO dvd_rental_client;

GRANT ALL ON TABLE film TO dvd_rental_admin;
```

grants all available privileges for this table.
Dangerous, but acceptable.

Roles

```
GRANT SELECT ON TABLE film TO dvd_rental_client;

GRANT ALL ON TABLE film TO dvd_rental_admin;

GRANT ALL ON TABLE film TO public;
```

grants all privileges on table to public, i.e. all roles in the database.
Dangerous and not acceptable!

Roles

```
GRANT SELECT ON TABLE film TO dvd_rental_client;

GRANT ALL ON TABLE film TO dvd_rental_admin;

GRANT ALL ON TABLE film TO public;

REVOKE ALL ON TABLE film FROM public;
```

remove dangerous privileges from public role

Roles

```
GRANT SELECT ON TABLE film TO dvd_rental_client;

GRANT ALL ON TABLE film TO dvd_rental_admin;

GRANT ALL ON TABLE film TO public;

REVOKE ALL ON TABLE film FROM public;

GRANT EXECUTE ON FUNCTION list_recent_films TO public;
```

allow all users to invoke function list_recent_films

Roles

```
GRANT SELECT ON TABLE film TO dvd_rental_client;

GRANT ALL ON TABLE film TO dvd_rental_admin;

GRANT ALL ON TABLE film TO public;

REVOKE ALL ON TABLE film FROM public;

GRANT EXECUTE ON FUNCTION list_recent_films TO public;

GRANT CREATE ON SCHEMA dvd_rental TO dvd_rental_admin;
```

allow dvd_rental_admin to manage objects in a schema

Roles

```
GRANT SELECT ON TABLE film TO dvd_rental_client;

GRANT ALL ON TABLE film TO dvd_rental_admin;

GRANT ALL ON TABLE film TO public;

REVOKE ALL ON TABLE film FROM public;

GRANT EXECUTE ON FUNCTION list_recent_films TO public;

GRANT CREATE ON SCHEMA dvd_rental TO dvd_rental_admin;

GRANT ALL ON ALL TABLES IN SCHEMA dvd_rental
TO dvd_rental_admin;
```

allow to perform DML on all tables in schema

Roles

```
GRANT SELECT ON TABLE film TO dvd_rental_client;

GRANT ALL ON TABLE film TO dvd_rental_admin;

GRANT ALL ON TABLE film TO public;

REVOKE ALL ON TABLE film FROM public;

GRANT EXECUTE ON FUNCTION list_recent_films TO public;

GRANT CREATE ON SCHEMA dvd_rental TO dvd_rental_admin;

GRANT ALL ON ALL TABLES IN SCHEMA dvd_rental
TO dvd_rental_admin;

GRANT CONNECT TO client_bob;
```

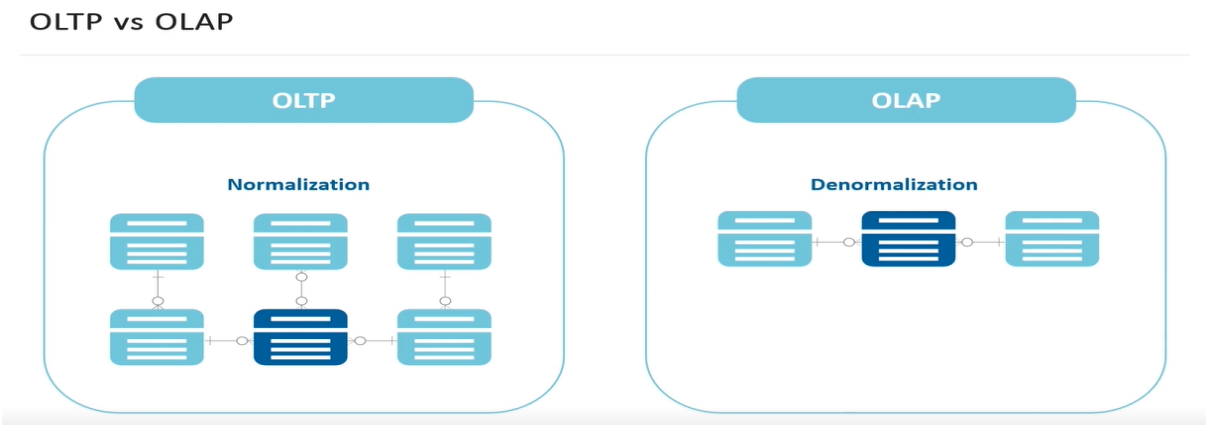
allow role client_bob to authenticate into the database.
***client_bob** role needs to be created with **LOGIN** option*

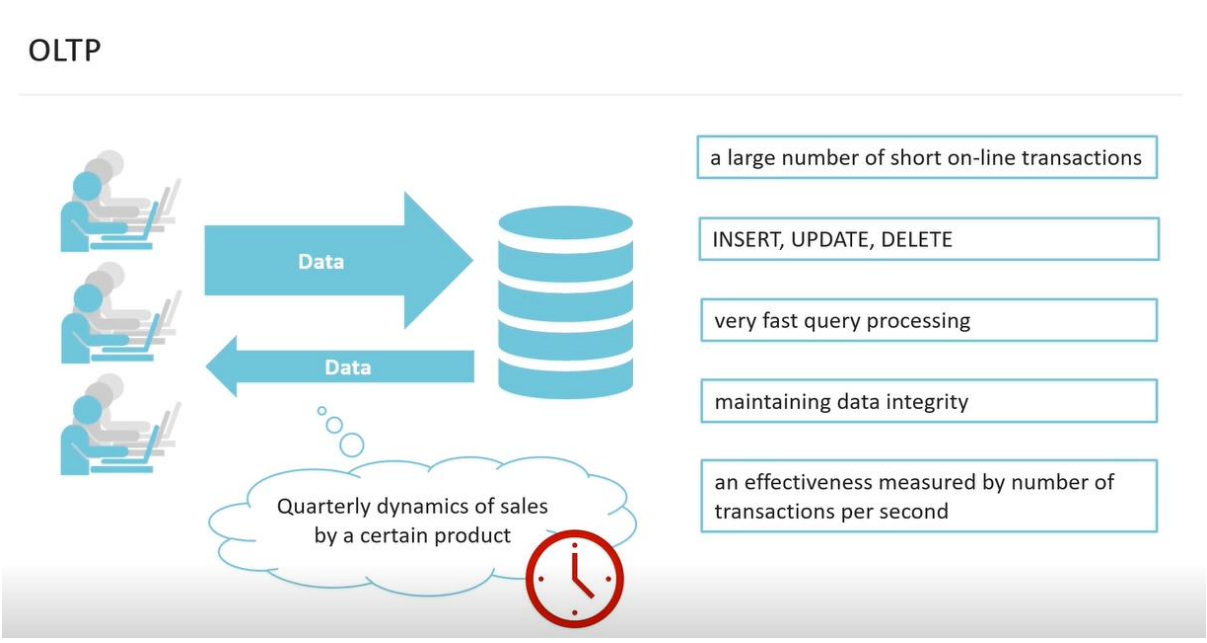
Choose the incorrect syntax:

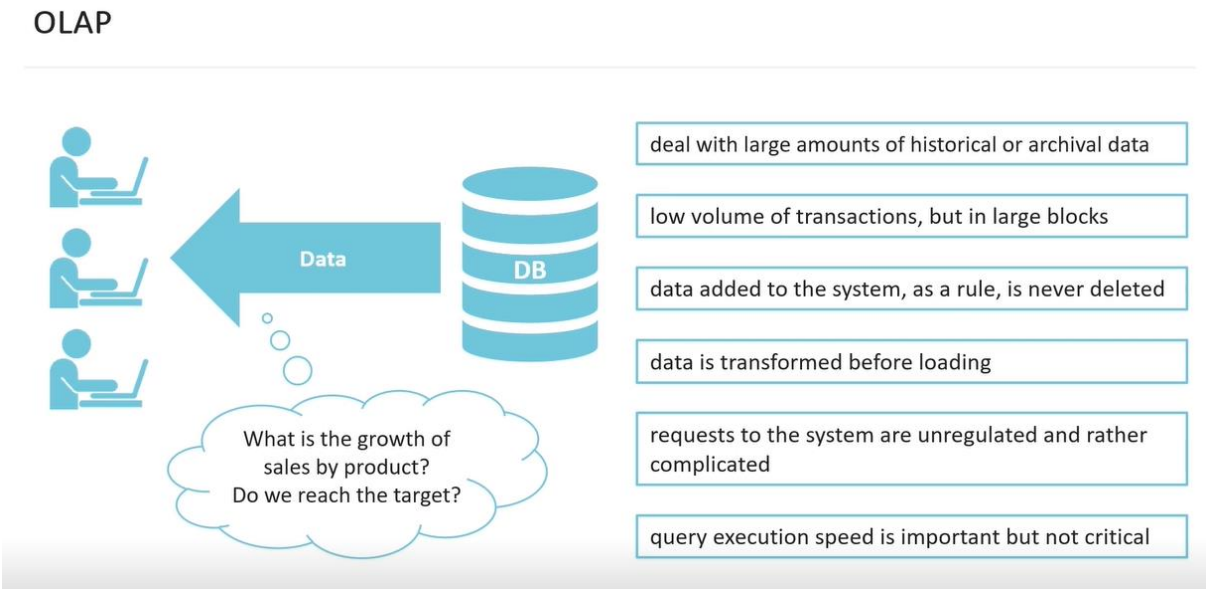
- [ ] GRANT ALL ON TABLE table_1 TO role_admin;
- [ ] GRANT ALL ON TABLE table_1 TO public;
- [x] GRANT role_admin TO public;
- [ ] GRANT ALL ON ALL TABLES IN SCHEMA schema_1 TO role_admin;
- [x] GRANT CONNECT TO USER client_bob;

✔

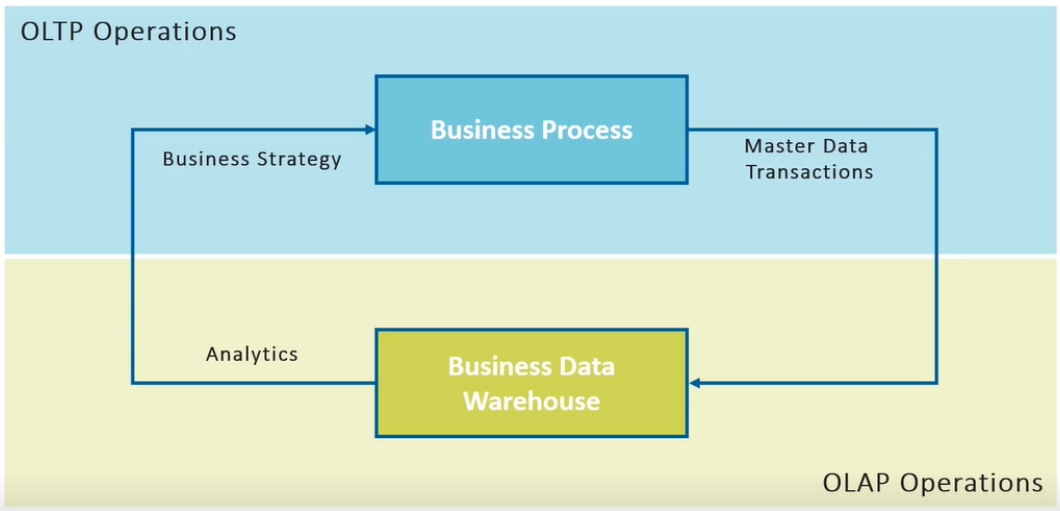
- **OLTP Vs OLAP :**


OLTP vs OLAP
OLTP
Normalization
OLAP
Denormalization


OLTP
Data
Data
Quarterly dynamics of sales by a certain product
a large number of short on-line transactions
INSERT, UPDATE, DELETE
very fast query processing
maintaining data integrity
an effectiveness measured by number of transactions per second


OLAP
Data
DB
What is the growth of sales by product? Do we reach the target?
deal with large amounts of historical or archival data
low volume of transactions, but in large blocks
data added to the system, as a rule, is never deleted
data is transformed before loading
requests to the system are unregulated and rather complicated
query execution speed is important but not critical

Different Information Worlds: OLTP and OLAP



OLTP and OLAP Comparison

	OLTP	OLAP
Source of Data	Operational data; OLTPs are the original source of data	Consolidation data: OLAP data comes from the various OLTP Database
Purpose of data	To control and run fundamental business tasks	To help planning, problem solving, and decision support
What the data	Reveals a snapshot of ongoing business processes	Multi-dimensional views of various kinds of business activities
Inserts and Updates	Short and fast inserts and updates initiated by end users	Periodic long-running batch jobs refresh the data
Queries	Relatively standardized and simple queries returning relatively few records	Often complex queries involving aggregations
Processing Speed	Typically very fast	Depends on the amount of data involved; batch data refreshes and complex queries may take many hours; query speed can be improved by creating indexes
Space Requirements	Can be relatively small if historical data is archived	Larger due to the existence of aggregation structures and history data; requires more indexes than OLTP
Database Design	Highly normalized with many tables	Typically de-normalized with fewer tables; use of star and/or snowflake schemas
Backup and Recovery	Backup religiously; operational data is critical to run the business, data loss is likely to entail significant monetary loss and legal liability	Instead of regular backups, some environments may consider simply reloading the OLTP data as a recovery method

- **Intro to DWH :**

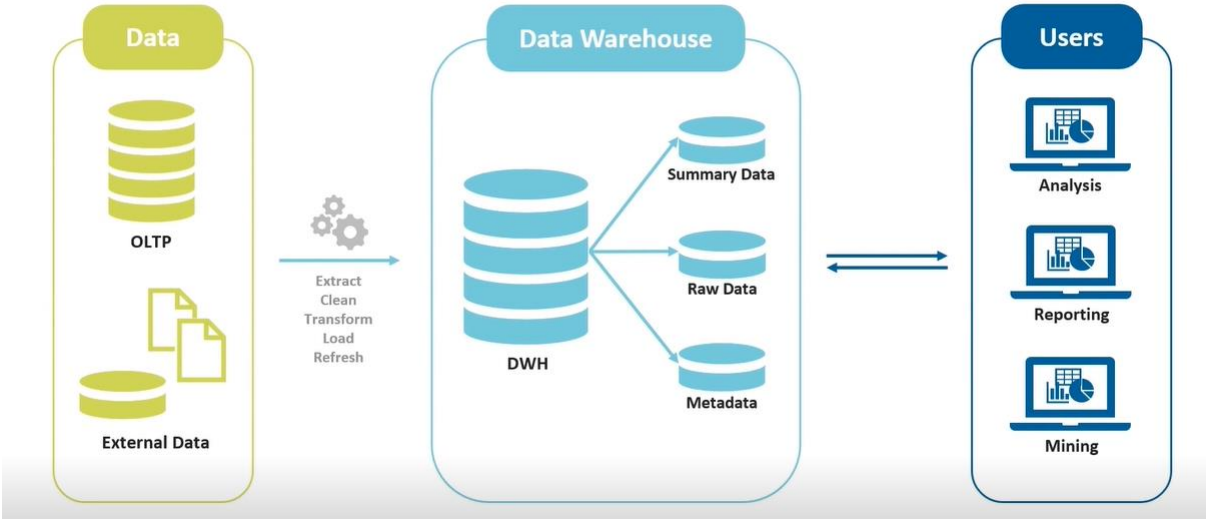
DWH Definition

A Data Warehouse is:

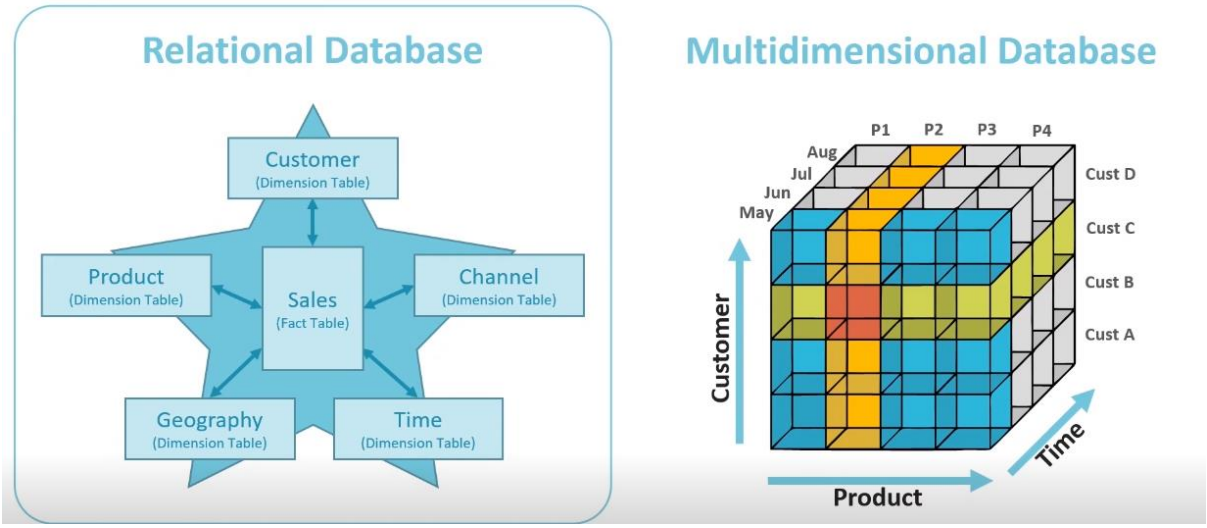
- Subject-Oriented
- Integrated
- Time-Variant
- Non-volatile

collection of data in support of management's decision making process

Typical Data Warehouse

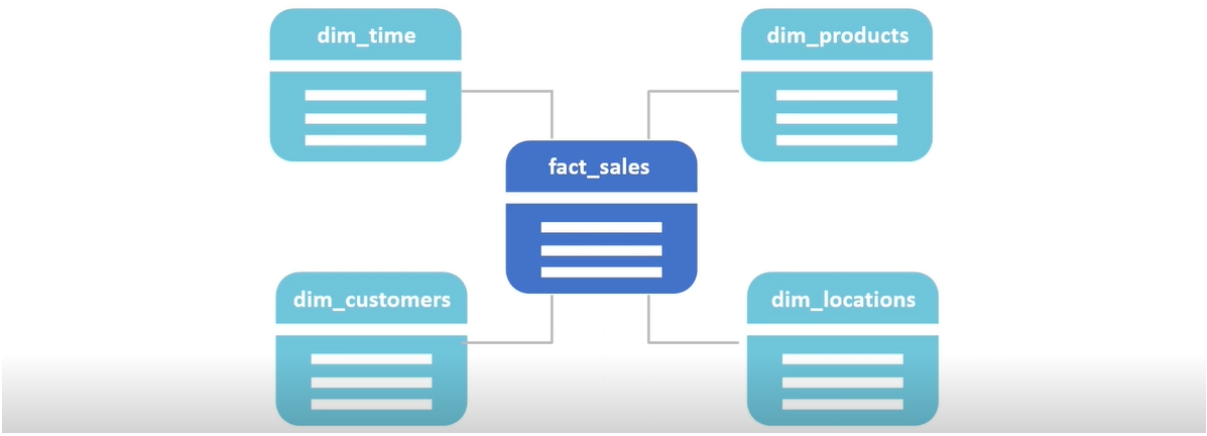


Data Warehouse Organization



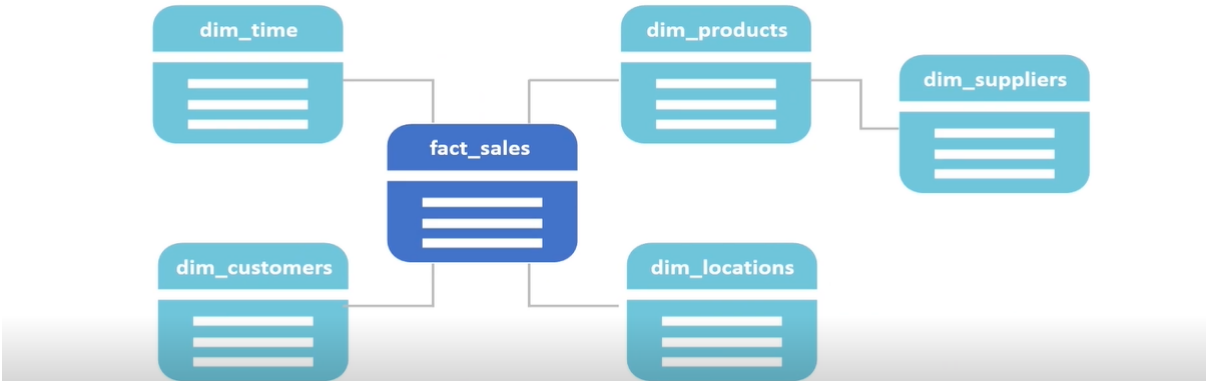
DWH Schemas: Star

The **Star** schema consists of one or more
fact tables referencing any number of dimension tables



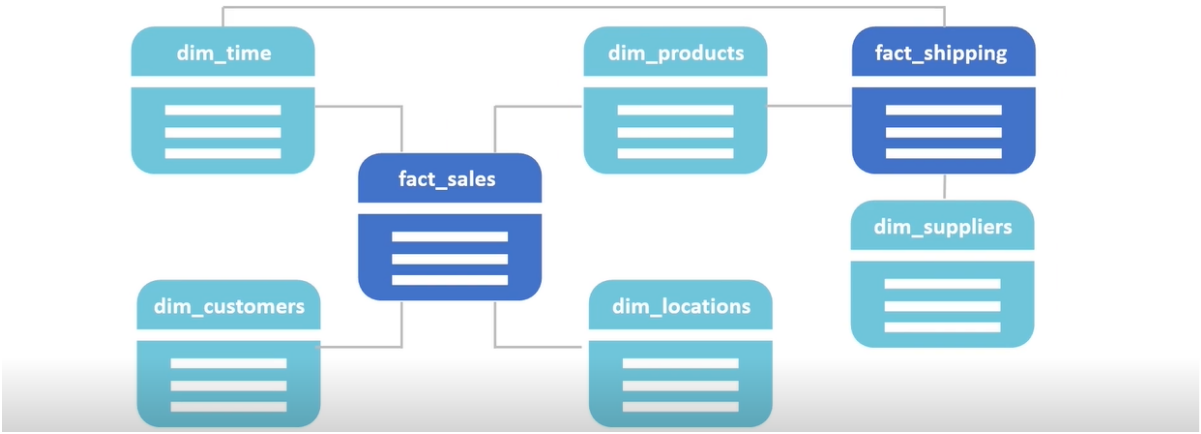
DWH Schemas: Snowflake

The **Snowflake** schema is a variation of the star schema,
featuring normalization of dimension tables.

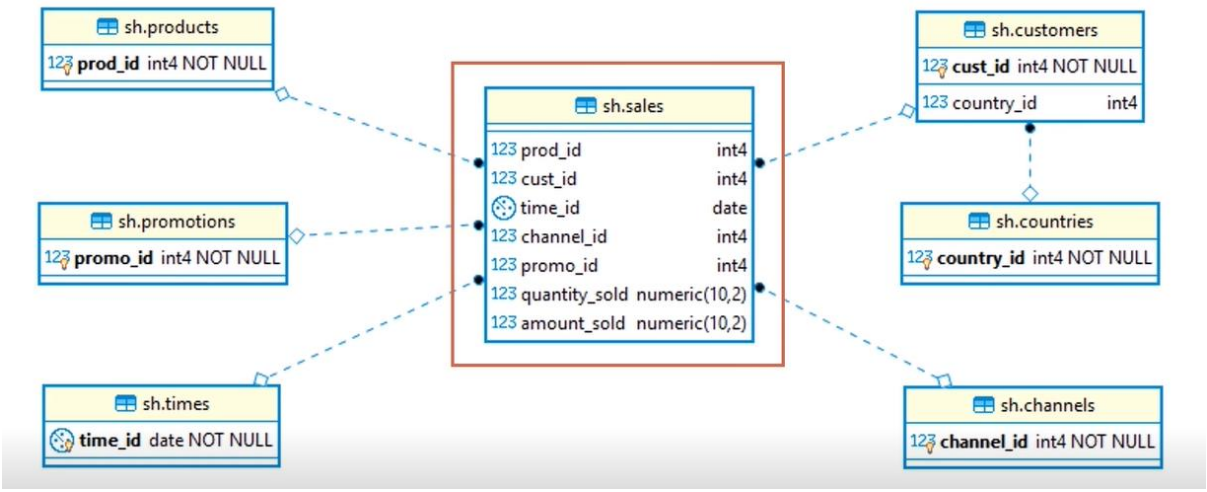


DWH Schemas: Fact constellation schema

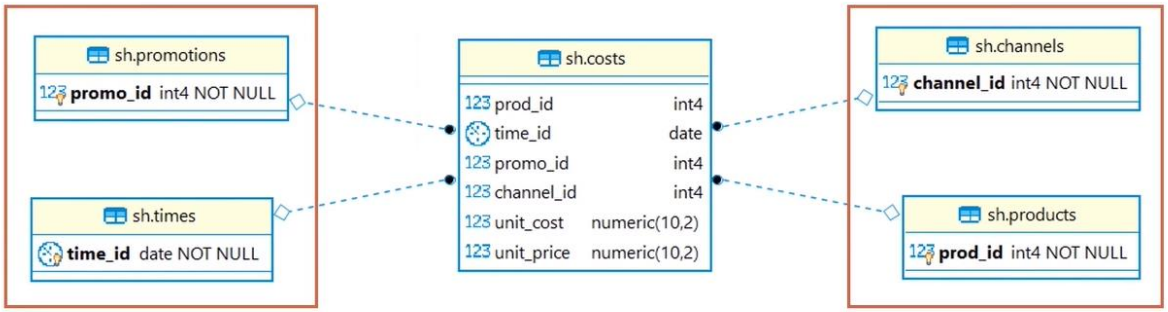
Fact constellation schema is collection of multiple **fact tables**
sharing dimension tables, viewed as a collection of stars



SH Sample Schema: Sales



SH Sample Schema: Costs



What is OLAP System?

- ○ System that facilitate and manage no-SQL applications, typically for transactions registration purposes with embedded reports about transactions history
- ○ System that facilitate and manage transaction-oriented applications, typically for reporting purposes with heavy analytical processing
- ○ System that facilitate and manage any database applications, which can be used as data source in data mining solution
- ◉ System that facilitate and manage analytical-oriented applications, typically for reporting purposes with heavy analytical processing

✔

What is not the goal(s) of DWH?

- ☐ Present the organization's information consistently
- ☑ Serve as operational data storage for OLTP system(s)
- ☐ Be acceptable by business community
- ☐ Be adaptive and resilient to change

✔

Fact table in star schema doesn't contain:

- ☐ fields which are storing foreign keys
- ☑ as many as possible of the dimension member's attribute fields
- ☐ fields which are storing the individual facts or measures

✔

Installation instruction

For further study of window functions we suggest you deploy another training database that allow you to turn knowledge into action.

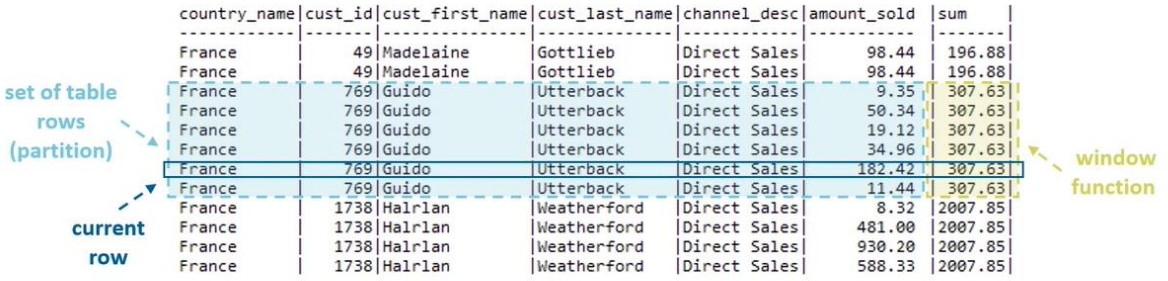
- Installation instruction can be found **here**.
- Database backup you can find **here**.

Please take the time to rewrite and analyze all queries from the video lessons to the database. Don't hesitate to contact mentors in case of any questions.

- **Window Functions :**

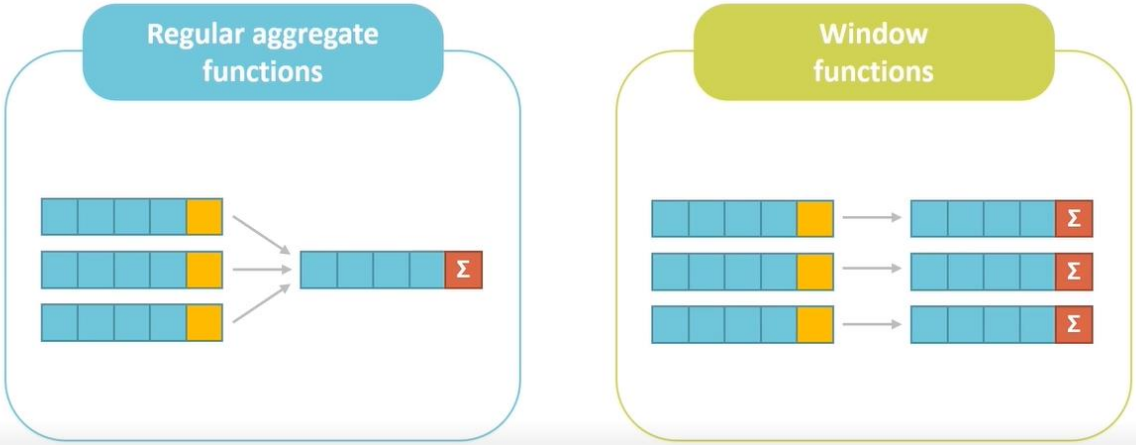
Window Functions

A *window function* performs a calculation across a *set of table rows* that are somehow related to the *current row*



country_name	cust_id	cust_first_name	cust_last_name	channel_desc	amount_sold	sum
France	49	Madelaine	Gottlieb	Direct Sales	98.44	196.88
France	49	Madelaine	Gottlieb	Direct Sales	98.44	196.88
France	769	Guido	Utterback	Direct Sales	9.35	307.63
France	769	Guido	Utterback	Direct Sales	50.34	307.63
France	769	Guido	Utterback	Direct Sales	19.12	307.63
France	769	Guido	Utterback	Direct Sales	34.96	307.63
France	769	Guido	Utterback	Direct Sales	182.42	307.63
France	769	Guido	Utterback	Direct Sales	11.44	307.63
France	1738	Halrlan	Weatherford	Direct Sales	8.32	2007.85
France	1738	Halrlan	Weatherford	Direct Sales	481.00	2007.85
France	1738	Halrlan	Weatherford	Direct Sales	930.20	2007.85
France	1738	Halrlan	Weatherford	Direct Sales	588.33	2007.85

Window Functions



Window Functions: Essential Concepts

SELECT	list of columns, window functions
FROM	table / joint tables / subquery
WHERE	filtering clause
GROUP BY	list of columns
HAVING	aggregation filtering clause

Partitions created; Window functions applied to each row in each partition

Window Functions: Essential Concepts

SELECT	list of columns, window functions
FROM	table / joint tables / subquery
WHERE	filtering clause
GROUP BY	list of columns
HAVING	aggregation filtering clause
ORDER BY	list of columns / window functions

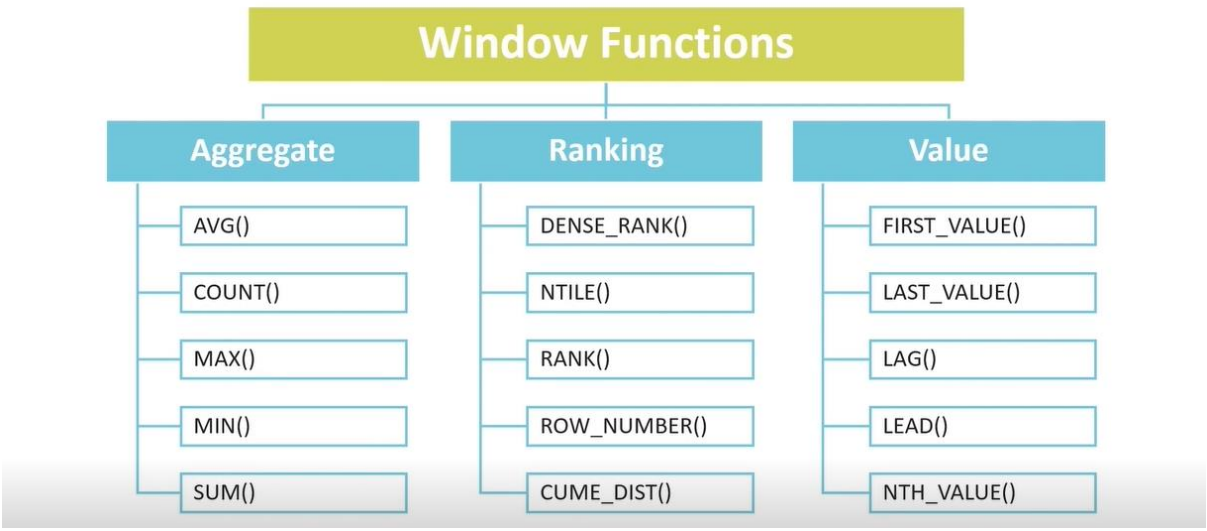
Result set partitions:

Partitions are created after the groups defined with GROUP BY clauses

Partitions are available to any aggregate results such as sums and averages

Partition divisions may be based upon any desired columns or expressions

Window Functions: Types



Window Function: Syntax

FUNCTION_NAME (column1) [FILTER (WHERE filter_clause)] OVER (window_definition)

PARTITION BY (column2)

ORDER BY (column2) [ASC | DESC | USING operator] [NULLS { FIRST | LAST }]

frame_clause

- ***Aggregate Window Functions :***

Aggregate Functions

```
SELECT    column_1,
          AGGREGATION_FUNCTION (column_2)     -- SUM | AVG | MIN | MAX | COUNT
FROM      table
GROUP BY column_1;
```

table_A	
name	amount
A	10
A	20
A	30

```
SELECT    name,
          SUM (amount)
FROM      table_A
GROUP BY name;
```

name	sum (amount)
A	60

Aggregate Functions

```
SELECT    column_1,
          column_2,
          AGGREGATION_FUNCTION ( [ DISTINCT ] column_2) OVER (PARTITION BY column_1)
FROM      table;
```

table_A	
name	amount
A	10
A	20
A	30

```
SELECT name,
       amount,
       SUM (amount) OVER (PARTITION BY name)
FROM   table_A;
```

name	amount	sum (amount)
A	10	60
A	20	60
A	30	60

Eg:

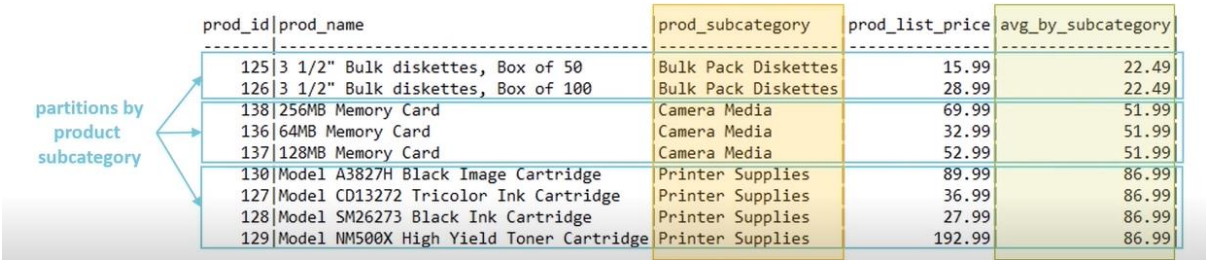
AVG() + PARTITION BY

```
SELECT   prod_id,
         prod_name,
         prod_subcategory,
         prod_list_price,
               AVG(prod_list_price)  OVER
FROM     sh.products
WHERE    prod_subcategory IN ('Bulk Pack Diskettes', 'Camera Media', 'Printer Supplies');
```

The **OVER** keyword signals a window function

AVG() + PARTITION BY

```
SELECT  prod_id,
        prod_name,
        prod_subcategory,
        prod_list_price,
        ROUND(AVG(prod_list_price)  OVER (PARTITION BY prod_subcategory), 2)
FROM    sh.products
WHERE   prod_subcategory IN ('Bulk Pack Diskettes', 'Camera Media', 'Printer Supplies');
```



prod_id	prod_name	prod_subcategory	prod_list_price	avg_by_subcategory
125	3 1/2" Bulk diskettes, Box of 50	Bulk Pack Diskettes	15.99	22.49
126	3 1/2" Bulk diskettes, Box of 100	Bulk Pack Diskettes	28.99	22.49
138	256MB Memory Card	Camera Media	69.99	51.99
136	64MB Memory Card	Camera Media	32.99	51.99
137	128MB Memory Card	Camera Media	52.99	51.99
130	Model A3827H Black Image Cartridge	Printer Supplies	89.99	86.99
127	Model CD13272 Tricolor Ink Cartridge	Printer Supplies	36.99	86.99
128	Model SM26273 Black Ink Cartridge	Printer Supplies	27.99	86.99
129	Model NM500X High Yield Toner Cartridge	Printer Supplies	192.99	86.99

SUM() + PARTITION BY

```
SELECT   cn.country_name,
         ch.channel_desc,
         SUM(amount_sold) AS sales$,
         SUM(SUM(amount_sold)) OVER (PARTITION BY cn.country_name) AS all_channels_sales$
FROM     sh.sales s
 JOIN    sh.products p ON p.prod_id = s.prod_id
 JOIN    sh.customers cust ON cust.cust_id = s.cust_id
 JOIN    sh.times t ON t.time_id = s.time_id
 JOIN    sh.channels ch ON ch.channel_id = s.channel_id
 JOIN    sh.countries cn ON cn.country_id = cust.country_id
WHERE    t.calendar_month_desc IN ('2000-09','2000-10')
AND      cn.country_iso_code IN ('AU', 'BR', 'CA', 'DE')
GROUP BY cn.country_name,
         ch.channel_desc
ORDER BY country_name;
```

country_name	channel_desc	sales$	all_channels_sales$
Australia	Direct Sales	97052.66	172636.47
Australia	Internet	12903.02	172636.47
Australia	Partners	62680.79	172636.47
Brazil	Direct Sales	8.46	1564.48
Brazil	Partners	1556.02	1564.48
Canada	Direct Sales	52323.40	103413.28
Canada	Internet	13547.60	103413.28
Canada	Partners	37542.28	103413.28
Germany	Direct Sales	274535.59	406232.18
Germany	Internet	36865.38	406232.18
Germany	Partners	94831.21	406232.18

Aggregate Function: Reporting

For each product category, find the region in which it had maximum sales

```
SELECT   prod_category,
         country_region,
         SUM(s.amount_sold) AS  sales,
         MAX(SUM(amount_sold)) OVER (PARTITION BY p.prod_category) AS max_reg_sales
FROM     sh.sales s
 JOIN    sh.products p ON p.prod_id =s.prod_id
 JOIN    sh.customers cust ON cust.cust_id = s.cust_id
 JOIN    sh.channels ch ON ch.channel_id = s.channel_id
 JOIN    sh.countries cn ON cn.country_id = cust.country_id
WHERE    time_id = TO_DATE('11-OCT-2001', 'DD-MON-YYYY')
GROUP BY prod_category,
         country_region;
```

prod_category	country_region	sales	max_reg_sales
Electronics	Americas	581.92	581.92
Hardware	Americas	925.93	925.93
Peripherals and Accessories	Americas	3084.48	4290.38
Peripherals and Accessories	Asia	2616.51	4290.38
Peripherals and Accessories	Europe	4290.38	4290.38
Peripherals and Accessories	Oceania	940.43	4290.38
Software/Other	Americas	4445.70	4445.70
Software/Other	Asia	1408.19	4445.70
Software/Other	Europe	3288.83	4445.70
Software/Other	Oceania	890.25	4445.70

Aggregate Function: Reporting

```
SELECT   prod_category,
         country_region,
         sales
FROM (
      SELECT   prod_category,
               country_region,
               SUM(s.amount_sold) AS  sales,
               MAX(SUM(amount_sold)) OVER (PARTITION BY p.prod_category) AS max_reg_sales
      FROM     sh.sales s
       JOIN    sh.products p ON p.prod_id =s.prod_id
       JOIN    sh.customers cust ON cust.cust_id = s.cust_id
       JOIN    sh.channels ch ON ch.channel_id = s.channel_id
       JOIN    sh.countries cn ON cn.country_id = cust.country_id
      WHERE    time_id = TO_DATE('11-OCT-2001', 'DD-MON-YYYY')
      GROUP BY prod_category,
               country_region
      ) tab
WHERE    sales = max_reg_sales;
```

prod_category	country_region	sales
Electronics	Americas	581.92
Hardware	Americas	925.93
Peripherals and Accessories	Europe	4290.38
Software/Other	Americas	4445.70

- **Ranking Window functions :**

ROW_NUMBER Function

```
SELECT   column_1,
         column_2,
         RANKING_FUNCTION ( ) OVER ( [PARTITION BY column_1] ORDER BY column_2 ASC | DESC)
FROM     table;
```

table_A

name	amount
A	10
A	20
A	20
A	30

```
SELECT name,
       amount,
       ROW_NUMBER () OVER (ORDER BY amount)
FROM   table_A;
```

name	amount	row_number
A	10	1
A	20	2
A	20	3
A	30	4

ROW_NUMBER

```
SELECT   ch.channel_desc,
         t.calendar_month_desc,
         TO_CHAR(TRUNC(SUM(amount_sold), -5), '9,999,999,999') AS sales$,
         ROW_NUMBER() OVER (ORDER BY TRUNC(SUM(amount_sold), -5) DESC) AS row_number
FROM     sh.sales s
 JOIN    sh.products p ON p.prod_id = s.prod_id
 JOIN    sh.customers cust ON cust.cust_id = s.cust_id
 JOIN    sh.times t ON t.time_id = s.time_id
 JOIN    sh.channels ch ON ch.channel_id = s.channel_id
 JOIN    sh.countries cn ON cn.country_id = cust.country_id
WHERE    calendar_month_desc IN ('2001-09', '2001-10')
GROUP BY channel_desc,
         calendar_month_desc;
```

channel_desc	calendar_month_desc	sales$	row_number
Direct Sales	2001-09	1,100,000	1
Direct Sales	2001-10	1,000,000	2
Internet	2001-10	700,000	3
Partners	2001-09	600,000	4
Partners	2001-10	600,000	5
Internet	2001-09	500,000	6

ROW_NUMBER + PARTITION BY

```
SELECT   ch.channel_desc,
         t.calendar_month_desc,
         TO_CHAR(TRUNC(SUM(amount_sold), -5), '9,999,999,999') AS sales$,
         ROW_NUMBER() OVER (PARTITION BY ch.channel_desc ORDER BY TRUNC(SUM(amount_sold), -5) DESC) AS row_number
FROM     sh.sales s
 JOIN    sh.products p ON p.prod_id = s.prod_id
 JOIN    sh.customers cust ON cust.cust_id = s.cust_id
 JOIN    sh.times t ON t.time_id = s.time_id
 JOIN    sh.channels ch ON ch.channel_id = s.channel_id
 JOIN    sh.countries cn ON cn.country_id = cust.country_id
WHERE    calendar_month_desc IN ('2001-09', '2001-10')
GROUP BY channel_desc,
         calendar_month_desc;
```

channel_desc	calendar_month_desc	sales$	row_number
Direct Sales	2001-09	1,100,000	1
Direct Sales	2001-10	1,000,000	2
Internet	2001-10	700,000	1
Internet	2001-09	500,000	2
Partners	2001-09	600,000	1
Partners	2001-10	600,000	2

ROW_NUMBER vs RANK vs DENSE_RANK

ROW_NUMBER

amount	row_number
10	1
20	2
20	3
30	4

RANK

amount	rank
10	1
20	2
20	2
30	4

DENSE_RANK

amount	dense_rank
10	1
20	2
20	2
30	3

RANK

```
SELECT   ch.channel_desc,
         TO_CHAR(SUM(amount_sold), '9,999,999,999') AS sales$,
         RANK () OVER (ORDER BY SUM(amount_sold)) AS default_rank,
         RANK () OVER (ORDER BY SUM(amount_sold) DESC NULLS LAST) AS custom_rank
FROM     sh.sales s
 JOIN    sh.products p ON p.prod_id = s.prod_id
 JOIN    sh.customers cust ON cust.cust_id = s.cust_id
 JOIN    sh.times t ON t.time_id = s.time_id
 JOIN    sh.channels ch ON ch.channel_id = s.channel_id
 JOIN    sh.countries cn ON cn.country_id = cust.country_id
WHERE    cn.country_iso_code = 'US'
AND      t.calendar_month_desc IN ('2000-09','2000-10')
GROUP BY ch.channel_desc;
```

```
channel_desc|sales$         |default_rank|custom_rank|
------------|--------------|------------|-----------|
Internet    |       261,278|           1|          3|
Partners    |       800,871|           2|          2|
Direct Sales|     1,320,497|           3|          1|
```

RANK on Multiple Expressions

```
SELECT   ch.channel_desc,
         t.calendar_month_desc,
         TO_CHAR(SUM(quantity_sold), '9,999,999,999') AS sales_count,
         RANK () OVER (ORDER BY calendar_month_desc, SUM(s.quantity_sold) DESC) AS col_rank
FROM     sh.sales s
 JOIN    sh.products p ON p.prod_id = s.prod_id
 JOIN    sh.customers cust ON cust.cust_id = s.cust_id
 JOIN    sh.times t ON t.time_id = s.time_id
 JOIN    sh.channels ch ON ch.channel_id = s.channel_id
 JOIN    sh.countries cn ON cn.country_id = cust.country_id
WHERE    ch.channel_desc != 'Tele Sales'
AND      t.calendar_month_desc IN ('2000-09', '2000-10')
GROUP BY ch.channel_desc,
         calendar_month_desc;
```

```
channel_desc|calendar_month_desc|sales_count   |col_rank|
------------|-------------------|--------------|--------|
Direct Sales|2000-09            |        11,995|       1|
Partners    |2000-09            |         6,165|       2|
Internet    |2000-09            |         1,887|       3|
Direct Sales|2000-10            |        12,584|       4|
Partners    |2000-10            |         7,508|       5|
Internet    |2000-10            |         1,450|       6|
```

RANK vs DENSE_RANK

```
SELECT   ch.channel_desc,
         t.calendar_month_desc,
         TO_CHAR(TRUNC(SUM(amount_sold), -5), '9,999,999,999') AS sales$,
         RANK () OVER (ORDER BY TRUNC(SUM(amount_sold), -5) DESC ) AS rank,
         DENSE_RANK () OVER (ORDER BY TRUNC(SUM(amount_sold), -5) DESC ) AS dense_rank
FROM     sh.sales s
 JOIN    sh.products p ON p.prod_id = s.prod_id
 JOIN    sh.customers cust ON cust.cust_id = s.cust_id
 JOIN    sh.times t ON t.time_id = s.time_id
 JOIN    sh.channels ch ON ch.channel_id = s.channel_id
 JOIN    sh.countries cn ON cn.country_id = cust.country_id
WHERE    ch.channel_desc != 'Tele Sales'
AND      t.calendar_month_desc IN ('2000-09', '2000-10')
GROUP BY ch.channel_desc,
         calendar_month_desc;
```

channel_desc	calendar_month_desc	sales$	rank	dense_rank
Direct Sales	2000-09	1,200,000	1	1
Direct Sales	2000-10	1,200,000	1	1
Partners	2000-09	600,000	3	2
Partners	2000-10	600,000	3	2
Internet	2000-09	200,000	5	3
Internet	2000-10	200,000	5	3

RANK + PARTITION BY

```
SELECT   ch.channel_desc,
         t.calendar_month_desc,
         TO_CHAR(SUM(amount_sold), '9,999,999,999') AS sales$,
         RANK () OVER (PARTITION BY ch.channel_desc ORDER BY SUM(amount_sold) DESC ) AS rank_by_channel
FROM     sh.sales s
 JOIN    sh.products p ON p.prod_id =s.prod_id
 JOIN    sh.customers cust ON cust.cust_id = s.cust_id
 JOIN    sh.times t ON t.time_id = s.time_id
 JOIN    sh.channels ch ON ch.channel_id = s.channel_id
 JOIN    sh.countries cn ON cn.country_id = cust.country_id
WHERE    t.calendar_month_desc IN ('2000-08', '2000-09', '2000-10', '2000-11')
AND      ch.channel_desc IN ('Direct Sales','Internet')
GROUP BY ch.channel_desc,
         calendar_month_desc;
```

channel_desc	calendar_month_desc	sales$	rank_by_channel
Direct Sales	2000-08	1,236,104	1
Direct Sales	2000-10	1,225,584	2
Direct Sales	2000-09	1,217,808	3
Direct Sales	2000-11	1,115,239	4
Internet	2000-11	284,742	1
Internet	2000-10	239,236	2
Internet	2000-09	228,241	3
Internet	2000-08	215,107	4

Per Group Ranking

```
SELECT   ch.channel_desc,
         t.calendar_month_desc,
         TO_CHAR(SUM(amount_sold), '9,999,999,999') AS sales$,
         RANK () OVER (PARTITION BY calendar_month_desc ORDER BY SUM(amount_sold) DESC) AS rank_within_month,
         RANK () OVER (PARTITION BY ch.channel_desc ORDER BY SUM(amount_sold) DESC) AS rank_within_channel
FROM     sh.sales s
 JOIN    sh.times t ON t.time_id = s.time_id
 JOIN    sh.channels ch ON ch.channel_id = s.channel_id
WHERE    t.calendar_month_desc IN ('2000-08', '2000-09', '2000-10', '2000-11')
AND      ch.channel_desc IN ('Direct Sales', 'Internet')
GROUP BY ch.channel_desc,
         calendar_month_desc
ORDER BY 1, 4, 5;
```

channel_desc	calendar_month_desc	sales$	rank_within_month	rank_within_channel
Direct Sales	2000-08	1,236,104	1	1
Direct Sales	2000-10	1,225,584	1	2
Direct Sales	2000-09	1,217,808	1	3
Direct Sales	2000-11	1,115,239	1	4
Internet	2000-11	284,742	2	1
Internet	2000-10	239,236	2	2
Internet	2000-09	228,241	2	3
Internet	2000-08	215,107	2	4

Per Group Ranking

```
SELECT   ch.channel_desc,
         t.calendar_month_desc,
         TO_CHAR(SUM(amount_sold), '9,999,999,999') AS sales$,
         RANK () OVER month_w AS rank_within_channel,
         RANK () OVER channel_w AS rank_within_channel
FROM     sh.sales s
 JOIN    sh.times t ON t.time_id = s.time_id
 JOIN    sh.channels ch ON ch.channel_id = s.channel_id
WHERE    t.calendar_month_desc IN ('2000-08', '2000-09', '2000-10', '2000-11')
AND      ch.channel_desc IN ('Direct Sales', 'Internet')
GROUP BY ch.channel_desc,
         calendar_month_desc
WINDOW   month_w AS (PARTITION BY calendar_month_desc ORDER BY SUM(amount_sold) DESC),
         channel_w AS (PARTITION BY ch.channel_desc ORDER BY SUM(amount_sold) DESC)
ORDER BY 1, 4, 5;
```

Treatment of NULLs

```
SELECT   t.time_id, sold,
         RANK () OVER (ORDER BY sold DESC NULLS LAST)  AS nlast_desc,
         RANK () OVER (ORDER BY sold DESC NULLS FIRST) AS nfirst_desc,
         RANK () OVER (ORDER BY sold ASC NULLS FIRST) AS nfirst,
         RANK () OVER (ORDER BY sold ASC NULLS LAST) AS nlast
FROM
  (SELECT time_id,
          SUM (amount_sold) AS sold
   FROM   sh.sales s
    JOIN  sh.products p ON (p.prod_id =s.prod_id)
    JOIN  sh.customers cust ON (cust.cust_id = s.cust_id)
    JOIN  sh.countries cn ON (cn.country_id = cust.country_id)
   WHERE  prod_name IN ('Envoy Ambassador', 'Mouse Pad')
   GROUP BY time_id
  ) tab
 RIGHT JOIN sh.times t ON tab.time_id = t.time_id
WHERE    t.calendar_year = 1999
AND      t.calendar_month_number = 1
ORDER BY sold DESC NULLS LAST;
```

time_id	sold	nlast_desc	nfirst_desc	nfirst	nlast
1999-01-14	25241.48	1	13	31	19
1999-01-21	24365.05	2	14	30	18
1999-01-10	22901.24	3	15	29	17
1999-01-20	16578.19	4	16	28	16
1999-01-16	15881.12	5	17	27	15
1999-01-30	15637.49	6	18	26	14
1999-01-17	13262.87	7	19	25	13
1999-01-25	13227.08	8	20	24	12
1999-01-03	9885.74	9	21	23	11
1999-01-28	4471.08	10	22	22	10
1999-01-27	3453.66	11	23	21	9
1999-01-23	925.45	12	24	20	8
1999-01-07	756.87	13	25	19	7
1999-01-08	571.80	14	26	18	6
1999-01-13	569.21	15	27	17	5
1999-01-02	316.87	16	28	16	4
1999-01-12	195.54	17	29	15	3
1999-01-26	92.96	18	30	14	2
1999-01-19	86.04	19	31	13	1
1999-01-15		20	1	1	20
1999-01-04		20	1	1	20
1999-01-06		20	1	1	20
1999-01-24		20	1	1	20
1999-01-11		20	1	1	20
1999-01-01		20	1	1	20
1999-01-09		20	1	1	20
1999-01-05		20	1	1	20

NTILE

```
SELECT   column_1,
         column_2,
         NTILE (x) OVER ( [PARTITION BY column_1] ORDER BY column_2 ASC | DESC)
FROM     table;
```

table_A	
name	amount
A	10
A	20
A	20
A	30

```
SELECT name,
       amount,
       NTILE (2) OVER (ORDER BY amount)
FROM   table_A;
```

name	amount	tile2
A	10	1
A	20	1
A	20	2
A	30	2

NTILE Function

```
SELECT   calendar_month_desc,
         TO_CHAR(SUM(s.amount_sold), '9,999,999,999') AS sales$,
         NTILE (4) OVER (ORDER BY SUM(amount_sold)) AS tile4,
         NTILE (5) OVER (ORDER BY SUM(amount_sold)) AS tile5,
         NTILE (15) OVER (ORDER BY SUM(amount_sold)) AS tile15
FROM     sh.sales s
 JOIN    sh.products p ON p.prod_id =s.prod_id
 JOIN    sh.customers cust ON cust.cust_id = s.cust_id
 JOIN    sh.times t ON t.time_id = s.time_id
 JOIN    sh.channels ch ON ch.channel_id = s.channel_id
WHERE    t.calendar_year = 2000
AND      prod_category = 'Electronics'
GROUP BY calendar_month_desc;
```

calendar_month_desc	sales$	tile4	tile5	tile15
2000-02	242,416	1	1	1
2000-01	257,286	1	1	2
2000-03	280,011	1	1	3
2000-06	315,951	2	2	4
2000-05	316,824	2	2	5
2000-04	318,106	2	2	6
2000-07	433,824	3	3	7
2000-08	477,833	3	3	8
2000-12	553,534	3	4	9
2000-10	652,225	4	4	10
2000-11	661,147	4	5	11
2000-09	691,449	4	5	12

CUME_DIST

```
SELECT   column_1,
         column_2,
         CUME_DIST() OVER ( [PARTITION BY column_1] ORDER BY column_2 ASC | DESC)
FROM     table;
```

CUME_DIST(x) = number of values in S coming before and including x in the specified order/ N

CUME_DIST

```
SELECT   column_1,
         column_2,
         CUME_DIST() OVER ( [PARTITION BY column_1] ORDER BY column_2 ASC | DESC)
FROM     table;
```

table_A	
name	amount
A	10
A	20
A	20
A	30

```
SELECT name,
       amount,
       CUME_DIST () OVER (ORDER BY amount)
FROM   table_A;
```

name	amount	cume_dist
A	10	1/4 = 0.25
A	20	3/4 = 0.75
A	20	3/4 = 0.75
A	30	4/4 = 1

CUME_DIST Function

```
SELECT    calendar_month_desc,
          p.prod_category,
          TO_CHAR(SUM(s.amount_sold), '9,999,999,999') AS sales$,
          CUME_DIST() OVER (PARTITION BY calendar_month_desc ORDER BY SUM(s.amount_sold)) AS cume_dist_prod_cat
FROM      sh.sales s
 JOIN     sh.products p ON p.prod_id = s.prod_id
 JOIN     sh.customers cust ON cust.cust_id = s.cust_id
 JOIN     sh.times t ON t.time_id = s.time_id
 JOIN     sh.channels ch ON ch.channel_id = s.channel_id
WHERE     t.calendar_month_desc IN ('2000-09', '2000-07', '2000-08')
GROUP BY calendar_month_desc,
          p.prod_category;
```

calendar_month_desc	prod_category	sales$	cume_dist_prod_cat
2000-07	Software/Other	295,419	0.2
2000-07	Hardware	323,381	0.4
2000-07	Photo	345,301	0.6
2000-07	Electronics	433,824	0.8
2000-07	Peripherals and Accessories	498,838	1
2000-08	Software/Other	316,150	0.2
2000-08	Hardware	338,568	0.4
2000-08	Photo	411,670	0.6
2000-08	Electronics	477,833	0.8
2000-08	Peripherals and Accessories	568,034	1
2000-09	Software/Other	245,110	0.2
2000-09	Photo	324,898	0.4
2000-09	Hardware	339,536	0.6
2000-09	Peripherals and Accessories	511,227	0.8
2000-09	Electronics	691,449	1

Ranking Function: Reporting

The following is a query which finds the 5 top-selling products for each product subcategory that contributes more than 20% of the sales within its product category.

```
SELECT    prod_category, prod_subcategory, p.prod_id, SUM(amount_sold) AS sales,
          SUM (SUM (amount_sold)) OVER (PARTITION BY prod_category) AS cat_sales,
          SUM (SUM (amount_sold)) OVER (PARTITION BY prod_subcategory) AS subcat_sales

FROM      sh.sales s
 JOIN     sh.products p ON p.prod_id = s.prod_id
 JOIN     sh.customers cust ON cust.cust_id = s.cust_id
 JOIN     sh.channels ch ON ch.channel_id = s.channel_id
 JOIN     sh.countries cn ON cn.country_id = cust.country_id
WHERE     time_id = TO_DATE ('11-OCT-2000','DD-MON-YYYY')
GROUP BY  prod_category,
          prod_subcategory,
          p.prod_id
ORDER BY  prod_category,
          prod_subcategory;
```

prod_category	prod_subcategory	prod_id	sales	cat_sales	subcat_sales
Peripherals and Accessories	Printer Supplies	127	2264.61	15976.16	15976.16
Peripherals and Accessories	Printer Supplies	129	11606.65	15976.16	15976.16
Peripherals and Accessories	Printer Supplies	128	2104.90	15976.16	15976.16
Software/Other	Bulk Pack Diskettes	126	563.55	7207.29	955.44
Software/Other	Bulk Pack Diskettes	125	391.89	7207.29	955.44
Software/Other	Recordable CDs	115	221.27	7207.29	1814.07
Software/Other	Recordable CDs	114	487.90	7207.29	1814.07
Software/Other	Recordable CDs	116	358.96	7207.29	1814.07
Software/Other	Recordable CDs	117	260.28	7207.29	1814.07
Software/Other	Recordable CDs	119	246.54	7207.29	1814.07
Software/Other	Recordable CDs	118	239.12	7207.29	1814.07
Software/Other	Recordable DVD Discs	124	1405.65	7207.29	4437.78
Software/Other	Recordable DVD Discs	123	3032.13	7207.29	4437.78

Ranking Function: Reporting

```
SELECT   prod_category, prod_subcategory, prod_id, sales
FROM (
      SELECT    prod_category, prod_subcategory, p.prod_id, SUM(amount_sold) AS sales,
                SUM (SUM (amount_sold)) OVER (PARTITION BY prod_category) AS cat_sales,
                SUM (SUM (amount_sold)) OVER (PARTITION BY prod_subcategory) AS subcat_sales,
                RANK() OVER (PARTITION BY prod_subcategory ORDER BY SUM(amount_sold) DESC) AS rank_in_line
      FROM      sh.sales s
       JOIN     sh.products p ON p.prod_id = s.prod_id
       JOIN     sh.customers cust ON cust.cust_id = s.cust_id
       JOIN     sh.channels ch ON ch.channel_id = s.channel_id
       JOIN     sh.countries cn ON cn.country_id = cust.country_id
      WHERE     time_id = TO_DATE ('11-OCT-2000','DD-MON-YYYY')
      GROUP BY  prod_category,
                prod_subcategory,
                p.prod_id
      ORDER BY  prod_category,
                prod_subcategory
      ) tab
WHERE   subcat_sales > 0.2*cat_sales
AND     rank_in_line <= 5;
```

prod_category	prod_subcategory	prod_id	sales
Peripherals and Accessories	Printer Supplies	128	2104.90
Peripherals and Accessories	Printer Supplies	129	11606.65
Peripherals and Accessories	Printer Supplies	127	2264.61
Software/Other	Recordable CDs	118	239.12
Software/Other	Recordable CDs	119	246.54
Software/Other	Recordable CDs	114	487.90
Software/Other	Recordable CDs	115	221.27
Software/Other	Recordable CDs	116	358.96
Software/Other	Recordable CDs	117	260.28
Software/Other	Recordable DVD Discs	123	3032.13
Software/Other	Recordable DVD Discs	124	1405.65

- **Offset Window Functions :**

LAG / LEAD

By default = 1

```
SELECT  column_1,
        column_2,
        LAG | LEAD (column_2, [OFFSET], [DEFAULT])
                OVER ([PARTITION BY column_1]  ORDER BY column_2 ASC | DESC )
FROM    table;
```

LAG / LEAD

```
SELECT   time_id,
         TO_CHAR(SUM(amount_sold), '9,999,999,999') AS sales$,
         TO_CHAR(LAG(SUM(amount_sold), 1) OVER (ORDER BY time_id), '9,999,999,999') AS LAG1,
         TO_CHAR(LEAD(SUM(amount_sold), 1) OVER (ORDER BY time_id), '9,999,999,999') AS LEAD1
FROM     sh.sales s
WHERE    time_id >= TO_DATE('10-OCT-2000', 'DD-MON-YYYY')
AND      time_id <= TO_DATE('17-OCT-2000', 'DD-MON-YYYY')
GROUP BY time_id;
```

time_id	sales$	lag1	lead1
2000-10-10	238,479		23,183
2000-10-11	23,183	238,479	24,616
2000-10-12	24,616	23,183	76,516
2000-10-13	76,516	24,616	29,795
2000-10-14	29,795	76,516	32,647
2000-10-15	32,647	29,795	63,257
2000-10-16	63,257	32,647	207,437
2000-10-17	207,437	63,257	

LAG / LEAD

```
SELECT   time_id,
         TO_CHAR(SUM(amount_sold), '9,999,999,999') AS sales$,
         TO_CHAR(LAG(SUM(amount_sold), 1) OVER (ORDER BY time_id), '9,999,999,999') AS LAG1,
         TO_CHAR(LEAD(SUM(amount_sold), 1) OVER (ORDER BY time_id), '9,999,999,999') AS LEAD1
FROM     sh.sales s
WHERE    time_id >= TO_DATE('10-OCT-2000', 'DD-MON-YYYY')
AND      time_id <= TO_DATE('17-OCT-2000', 'DD-MON-YYYY')
GROUP BY time_id;
```

time_id	sales$	lag1	lead1
2000-10-10	238,479		23,183
2000-10-11	23,183	238,479	24,616
2000-10-12	24,616	23,183	76,516
2000-10-13	76,516	24,616	29,795
2000-10-14	29,795	76,516	32,647
2000-10-15	32,647	29,795	63,257
2000-10-16	63,257	32,647	207,437
2000-10-17	207,437	63,257	

FIRST_VALUE / LAST_VALUE

will be retrieving a value after evaluating the first (last) row of a result set's sorted partition

```
SELECT  column_1,
        column_2,
        FIRST_VALUE | LAST_VALUE  (column_2)
                OVER ([PARTITION BY column_1]  ORDER BY column_2 ASC | DESC )
FROM    table;
```

FIRST_VALUE

```
SELECT    channel_desc,
          calendar_month_number,
          SUM(amount_sold) AS sales$,
          FIRST_VALUE(SUM(amount_sold)) OVER (PARTITION BY calendar_month_number ORDER BY SUM(amount_sold))
          AS min_sales_month
FROM      sh.sales s
 JOIN     sh.times t ON t.time_id = s.time_id
 JOIN     sh.channels ch ON ch.channel_id = s.channel_id
WHERE     calendar_month_number IN (1, 2, 3, 4)
GROUP BY channel_desc,
          calendar_month_number
ORDER BY 2;
```

channel_desc	calendar_month_number	sales$	min_sales_month
Internet	1	679299.05	679299.05
Partners	1	2092590.24	679299.05
Direct Sales	1	5703058.55	679299.05
Internet	2	1018813.31	1018813.31
Partners	2	2410599.08	1018813.31
Direct Sales	2	5522651.25	1018813.31
Tele Sales	3	53395.37	53395.37
Internet	3	1069167.19	53395.37
Partners	3	1876050.63	53395.37
Direct Sales	3	4680793.47	53395.37
Tele Sales	4	42712.02	42712.02
Internet	4	981296.96	42712.02
Partners	4	2091816.42	42712.02
Direct Sales	4	4578892.91	42712.02

LAST_VALUE

```
SELECT    channel_desc,
          calendar_month_number,
          SUM(amount_sold) AS sales$,
          LAST_VALUE(SUM(amount_sold)) OVER (PARTITION BY calendar_month_number ORDER BY SUM(amount_sold))
          AS max_sales_month
FROM      sh.sales s
 JOIN     sh.times t ON t.time_id = s.time_id
 JOIN     sh.channels ch ON ch.channel_id = s.channel_id
WHERE     calendar_month_number IN (1, 2, 3, 4)
GROUP BY channel_desc,
          calendar_month_number
ORDER BY 2;
```

channel_desc	calendar_month_number	sales$	max_sales_month
Internet	1	679299.05	679299.05
Partners	1	2092590.24	2092590.24
Direct Sales	1	5703058.55	5703058.55
Internet	2	1018813.31	1018813.31
Partners	2	2410599.08	2410599.08
Direct Sales	2	5522651.25	5522651.25
Tele Sales	3	53395.37	53395.37
Internet	3	1069167.19	1069167.19
Partners	3	1876050.63	1876050.63
Direct Sales	3	4680793.47	4680793.47
Tele Sales	4	42712.02	42712.02
Internet	4	981296.96	981296.96
Partners	4	2091816.42	2091816.42
Direct Sales	4	4578892.91	4578892.91

LAST_VALUE: Unexpected results

```
SELECT x,
LAST_VALUE(x) OVER (ORDER BY x),
LAST_VALUE(x) OVER (ORDER BY x DESC)
FROM GENERATE_SERIES(1, 10) AS x;
```

```
 x |last_value|last_value|
---|----------|----------|
 1|         1|         1|
 2|         2|         2|
 3|         3|         3|
 4|         4|         4|
 5|         5|         5|
 6|         6|         6|
 7|         7|         7|
 8|         8|         8|
 9|         9|         9|
10|        10|        10|
```

```
SELECT x,
ARRAY_AGG(x) OVER (ORDER BY x),
ARRAY_AGG(x) OVER (ORDER BY x DESC)
FROM GENERATE_SERIES(1, 10) AS x;
```

```
 x |array_agg             |array_agg              |
---|----------------------|-----------------------|
 1|{1}                   |{10,9,8,7,6,5,4,3,2,1} |
 2|{1,2}                 |{10,9,8,7,6,5,4,3,2}   |
 3|{1,2,3}               |{10,9,8,7,6,5,4,3}     |
 4|{1,2,3,4}             |{10,9,8,7,6,5,4}       |
 5|{1,2,3,4,5}           |{10,9,8,7,6,5}         |
 6|{1,2,3,4,5,6}         |{10,9,8,7,6}           |
 7|{1,2,3,4,5,6,7}       |{10,9,8,7}             |
 8|{1,2,3,4,5,6,7,8}     |{10,9,8}               |
 9|{1,2,3,4,5,6,7,8,9}   |{10,9}                 |
10|{1,2,3,4,5,6,7,8,9,10}|{10}                   |
```

Choose the rigth statement(s):

- [ ] DENSE_RANK leaves gaps in ranking sequence if there is a tie between the ranks of the preceding records
- [x] DENSE_RANK does not skip any ranks if there is a tie between the ranks of the preceding records
- [ ] RANK assigns a unique number (starting from 1, as defined by order) to each row within the partition
- [x] ROW_NUMBER assigns a unique number (starting from 1, as defined by order) to each row within the partition

✔

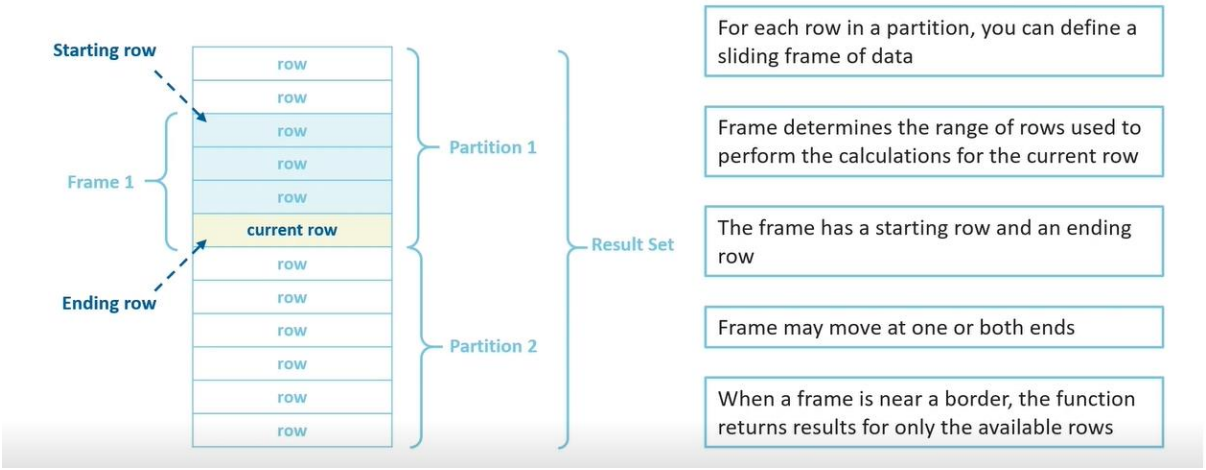
This function returns a value evaluated against the first row in a sorted partition of a result set

- ( ) LAG
- ( ) LEAD
- (•) FIRST_VALUE
- ( ) RANK

✔

- **Window Frames :**

Sliding Frame



Window Function: Syntax

```
FUNCTION_NAME (column1)  [ FILTER (WHERE filter_clause) ]  OVER (window_definition)

PARTITION BY (column2)

ORDER BY (column2)  [ ASC | DESC | USING operator ]  [ NULLS { FIRST | LAST } ]

frame_clause
```

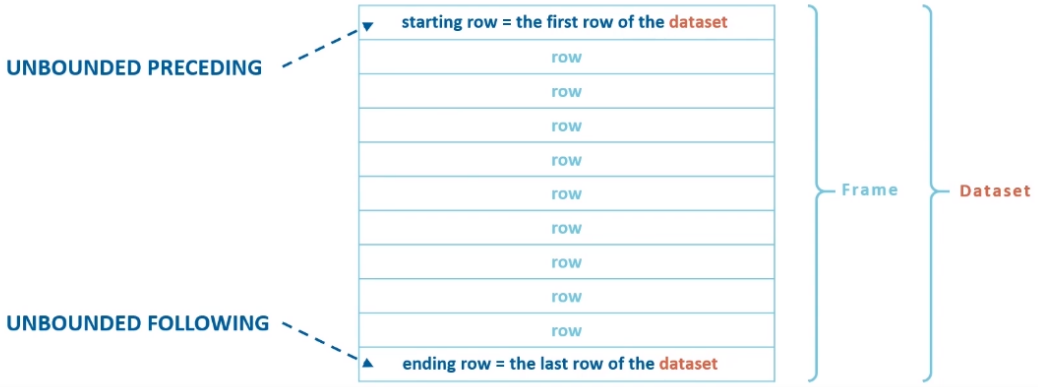
Window Function: frame clause syntax

```
RANGE | ROWS | GROUPS  BETWEEN  frame_start  AND  frame_end

UNBOUNDED PRECEDING
offset PRECEDING
CURRENT ROW
offset FOLLOWING
UNBOUNDED FOLLOWING
```

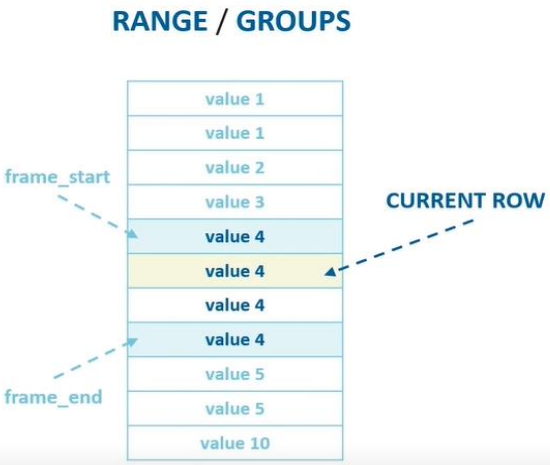
```
RANGE | ROWS | GROUPS  BETWEEN  2 PRECEDING  AND  3 PRECEDING
```

- **Unbounded precedings / following :**

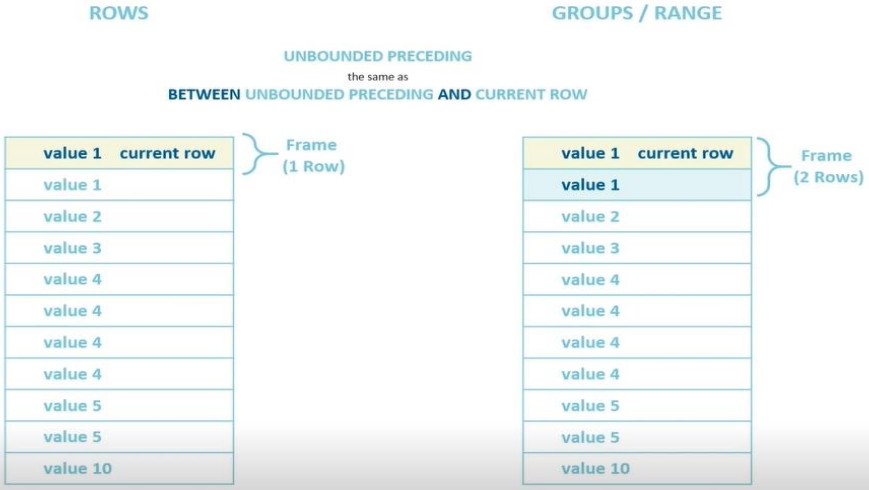
UNBOUNDED PRECEDING / UNBOUNDED FOLLOWING



UNBOUNDED PRECEDING / UNBOUNDED FOLLOWING and CURRENT ROW

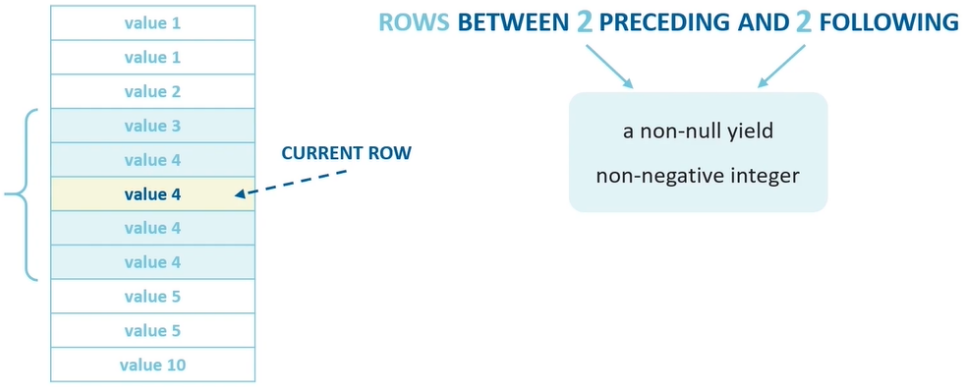


UNBOUNDED PRECEDING / UNBOUNDED FOLLOWING and CURRENT ROW

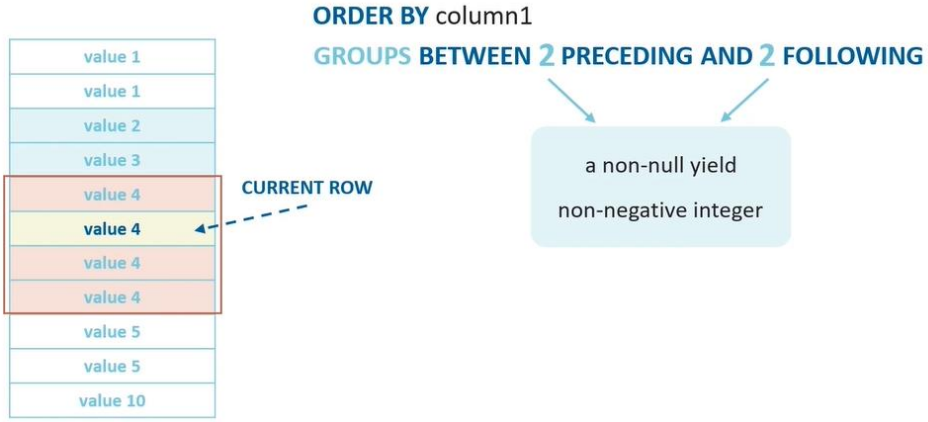


- **Offset preceding / following :**

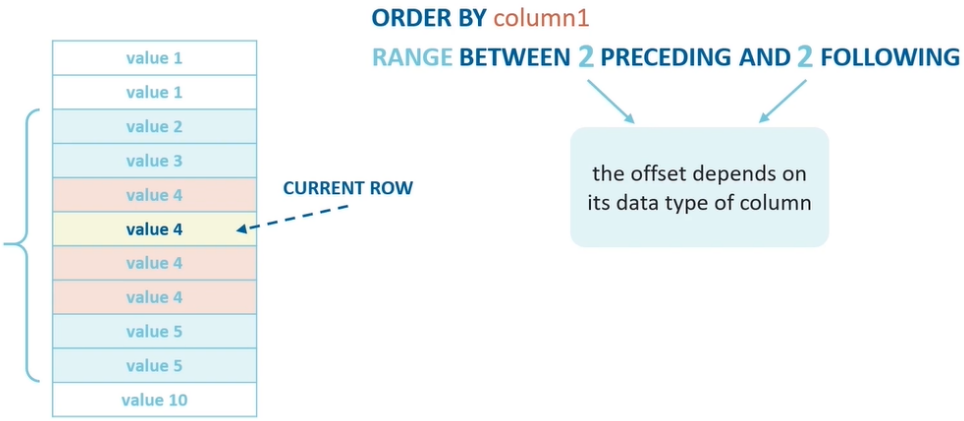
ROWS mode



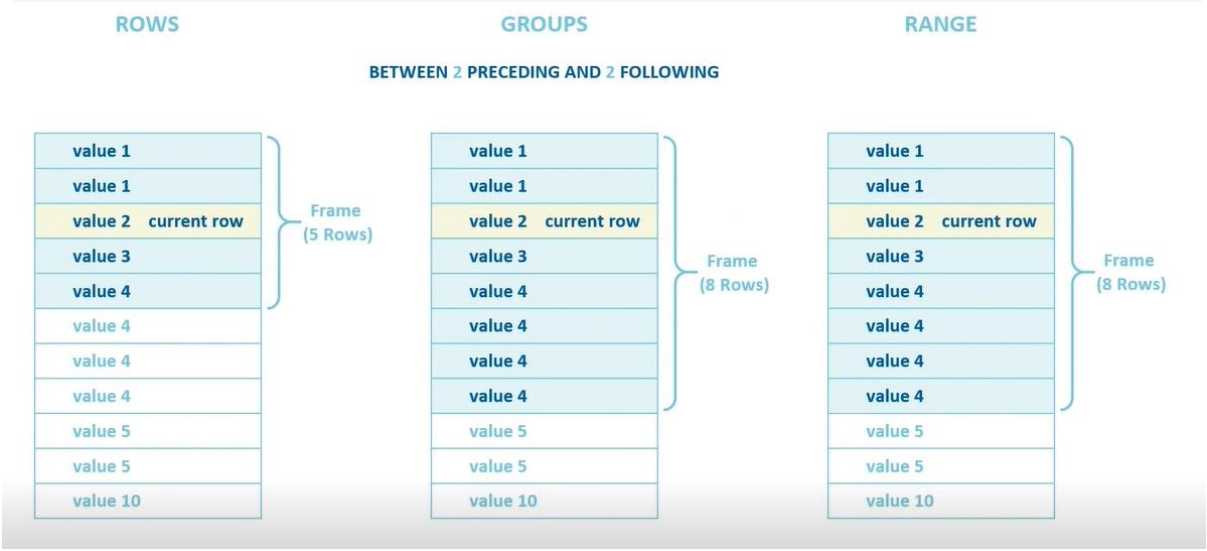
GROUPS mode



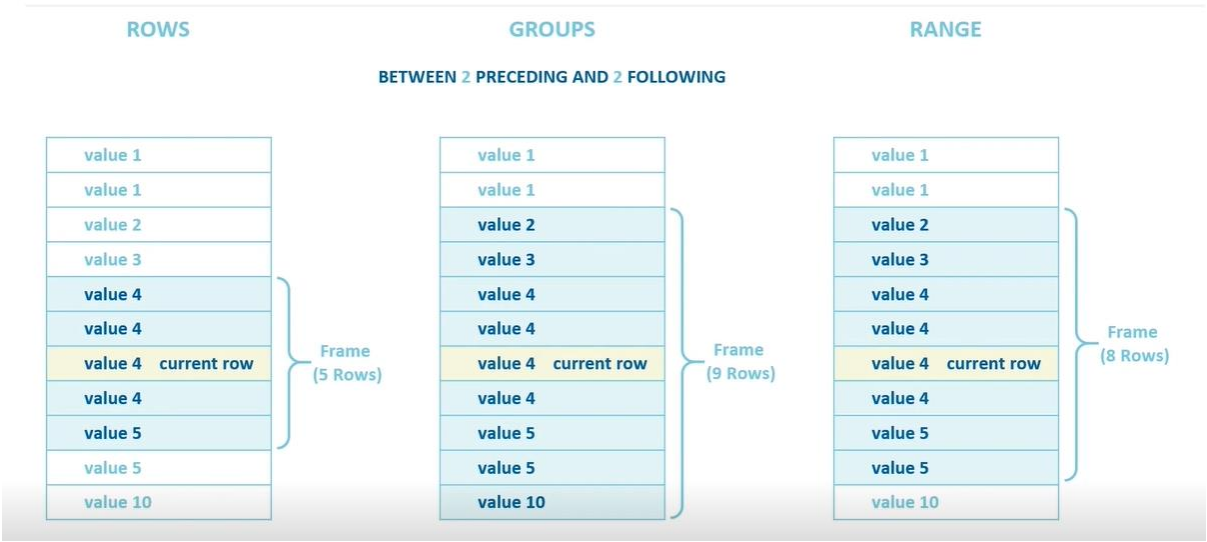
RANGE mode



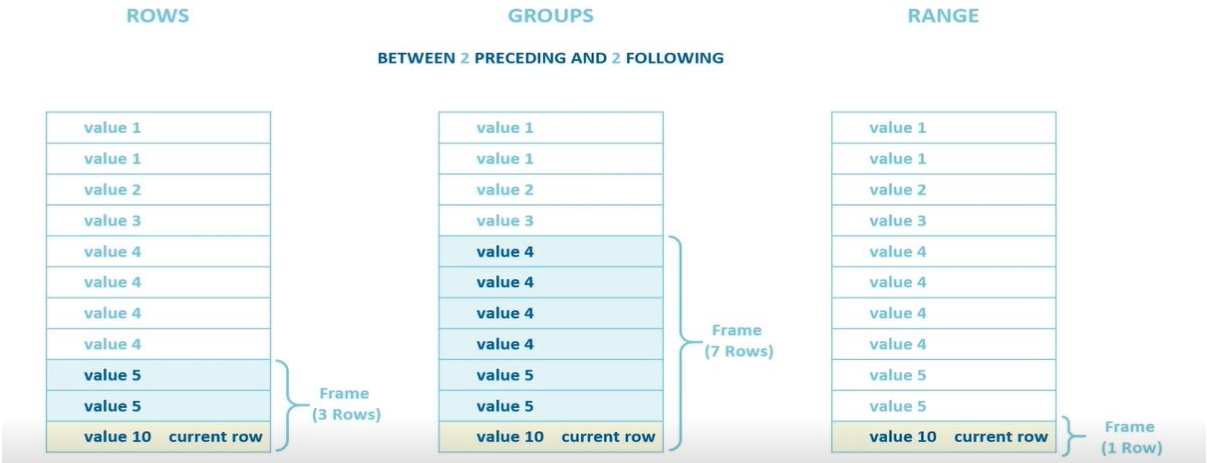
ROWS vs GROUPS vs RANGE



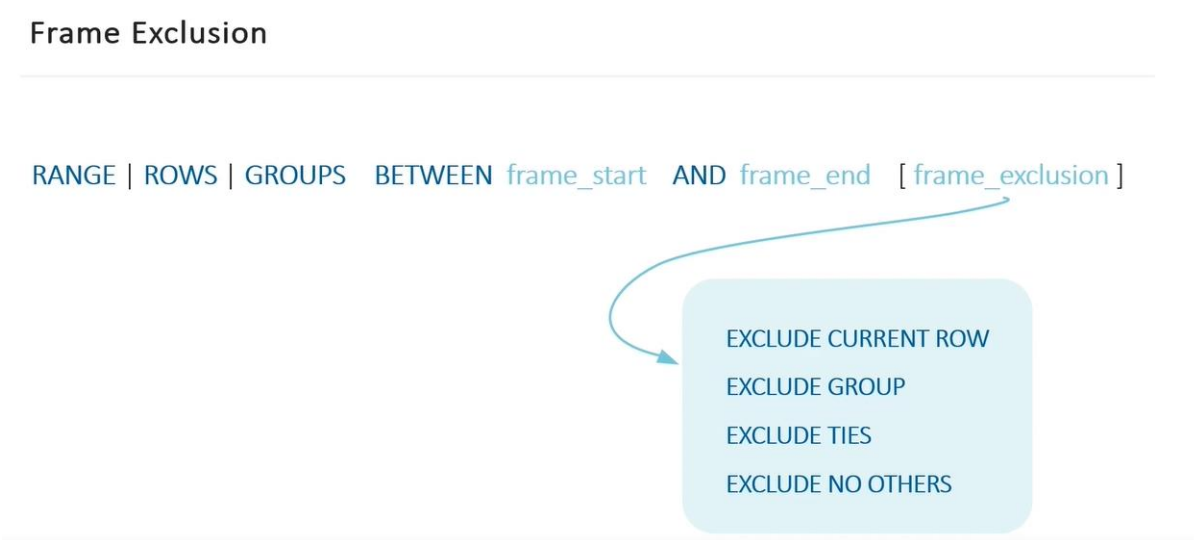
ROWS vs GROUPS vs RANGE



ROWS vs GROUPS vs RANGE



- **Frame Exclusion :**



Examples :

Cumulative amount_sold

```
SELECT    c.cust_id, t.calendar_quarter_desc,
          SUM(amount_sold) AS q_sales,
          SUM(SUM(amount_sold)) OVER (PARTITION BY c.cust_id ORDER BY t.calendar_quarter_desc
                                      RANGE BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW) AS cum_sales
FROM      sh.sales s
 JOIN     sh.customers c ON c.cust_id = s.cust_id
 JOIN     sh.times t ON t.time_id = s.time_id
WHERE     c.cust_id IN (2595, 9646, 11111)
AND       t.calendar_year = 2000
GROUP BY  c.cust_id,
          t.calendar_quarter_desc
ORDER BY  c.cust_id,
          t.calendar_quarter_desc;
```

cust_id	calendar_quarter_desc	q_sales	cum_sales
2595	2000-01	659.92	659.92
2595	2000-02	224.79	884.71
2595	2000-03	313.90	1198.61
2595	2000-04	6015.08	7213.69
9646	2000-01	1337.09	1337.09
9646	2000-02	185.67	1522.76
9646	2000-03	203.86	1726.62
9646	2000-04	458.29	2184.91
11111	2000-01	43.18	43.18
11111	2000-02	33.33	76.51
11111	2000-03	579.73	656.24
11111	2000-04	307.58	963.82

- **Nth Value function :**

NTH_VALUE

returns a value from the n row in an ordered partition of a result set

```
SELECT  column_1,
        column_2,
        NTH_VALUE (column_2, offset)
                OVER ([PARTITION BY column_1]
                        ORDER BY column_2 ASC | DESC )
                        frame_clause
                        )
FROM    table;
```

NTH_VALUE

```
SELECT    c.cust_id,
          t.calendar_quarter_desc,
          SUM(amount_sold) AS q_sales,
          FIRST_VALUE(SUM(amount_sold)) OVER w AS q1,
          NTH_VALUE(SUM(amount_sold), 2) OVER w AS q2,
          NTH_VALUE(SUM(amount_sold), 3) OVER w AS q3,
          LAST_VALUE(SUM(amount_sold)) OVER w AS q4
FROM      sh.sales s
 JOIN     sh.customers c ON c.cust_id = s.cust_id
 JOIN     sh.times t ON t.time_id = s.time_id
WHERE     t.calendar_year = 2000
AND       c.cust_id IN (2595, 9646, 11111)
GROUP BY  c.cust_id,
          t.calendar_quarter_desc,
          t.calendar_quarter_number
WINDOW    w AS (PARTITION BY c.cust_id ORDER BY c.cust_id, t.calendar_quarter_desc
                ROWS BETWEEN UNBOUNDED PRECEDING AND UNBOUNDED FOLLOWING)
ORDER BY  c.cust_id,
          t.calendar_quarter_desc;
```

cust_id	calendar_quarter_desc	q_sales	q1	q2	q3	q4
2595	2000-01	659.92	659.92	224.79	313.90	6015.08
2595	2000-02	224.79	659.92	224.79	313.90	6015.08
2595	2000-03	313.90	659.92	224.79	313.90	6015.08
2595	2000-04	6015.08	659.92	224.79	313.90	6015.08
9646	2000-01	1337.09	1337.09	185.67	203.86	458.29
9646	2000-02	185.67	1337.09	185.67	203.86	458.29
9646	2000-03	203.86	1337.09	185.67	203.86	458.29
9646	2000-04	458.29	1337.09	185.67	203.86	458.29
11111	2000-01	43.18	43.18	33.33	579.73	307.58
11111	2000-02	33.33	43.18	33.33	579.73	307.58
11111	2000-03	579.73	43.18	33.33	579.73	307.58
11111	2000-04	307.58	43.18	33.33	579.73	307.58