Advanced SQL

One thing we should know before we start …

“Big data isn’t about bits, it’s about talent”

- Douglas Merrill

DigitalSkola
Uncover The World Of Digital Skills With Us

Table of Content

What will We Learn Today?

1. Date Functions
2. Primary key, auto increment and foreign key
3. Database normalization
4. Join and Merge
5. Subquery
6. Bonus!!!

Date Functions

SQL date functions allow you to manipulate date and/or time data effectively.

Functions:

1. CURRENT_DATE: Return the current date
2. CURRENT_TIME: Return the current time
3. EXTRACT: Get timestamp subfield
4. DATE_TRUNC: Truncate to specified precision

Date Functions

- date + integer → date
- date - date → integer
- date + interval → timestamp
- date + time → timestamp
- interval + interval → interval
- timestamp + interval → timestamp
- time + interval → time
- date - integer → date
- date - interval → timestamp
- time - time → interval
- time - interval → time
- timestamp - interval → timestamp
- interval - interval → interval
- timestamp - timestamp → interval
- interval * double precision → interval
- interval / double precision → interval

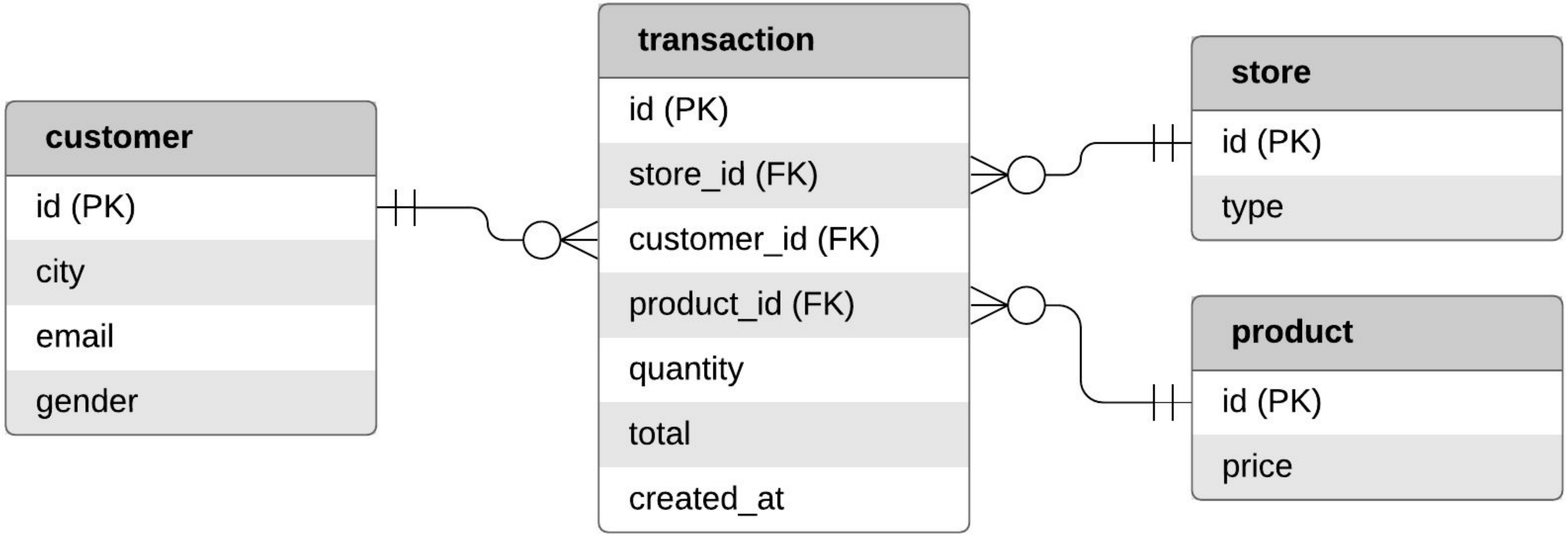
Key

An attribute or set of attributes which helps you to identify a row in a table. It allows you to find the relation between two tables. Key helps you uniquely identify a row in a table by combination of one or more columns in that table.

Why we need key?

1. To helps us identify any row of data in a table. In a real world data, a table could contains a million of records. Using a key could help us to identify if the records is duplicated or not.
2. Allows us to establish relationship between tables.

Key



customer
id (PK)
city
email
gender

transaction
id (PK)
store_id (FK)
customer_id (FK)
product_id (FK)
quantity
total
created_at

store
id (PK)
type

product
id (PK)
price

Key

An attribute or set of attributes which helps you to identify a row in a table. It allows you to find the relation between two tables. Key helps you uniquely identify a row in a table by combination of one or more columns in that table.

Why we need key?

1. To helps us identify any row of data in a table. In a real world data, a table could contains a million of records. Using a key could help us to identify if the records is duplicated or not.
2. Allows us to establish relationship between tables.

Key

- **Super key:** A set of columns in a table that can uniquely identify each record in a table. Super key is a superset of candidate key.
- **Candidate key:** A minimal set of columns which can uniquely identify each record in a table.
- **Primary key:** The most appropriate candidate key that is to become the main key.
- **Composite key:** Key that consists of two or more attributes that uniquely identify any records in a table. An individual key which together form a composite key cannot act as a key individually/independently.
- **Foreign key:** A column that creates a relationship between two tables.
- **Secondary/alternative key:** The candidate key which are not selected as primary key.
- **Non-key attributes:** The attributes that are other than candidate key of the table.
- **Non-prime attributes:** The attributes that are other than primary key of the table.

Key

employee_id	name	phone	age
1	A	12345	22
2	B	12346	21
3	C	12347	22

Primary Key = employee_id

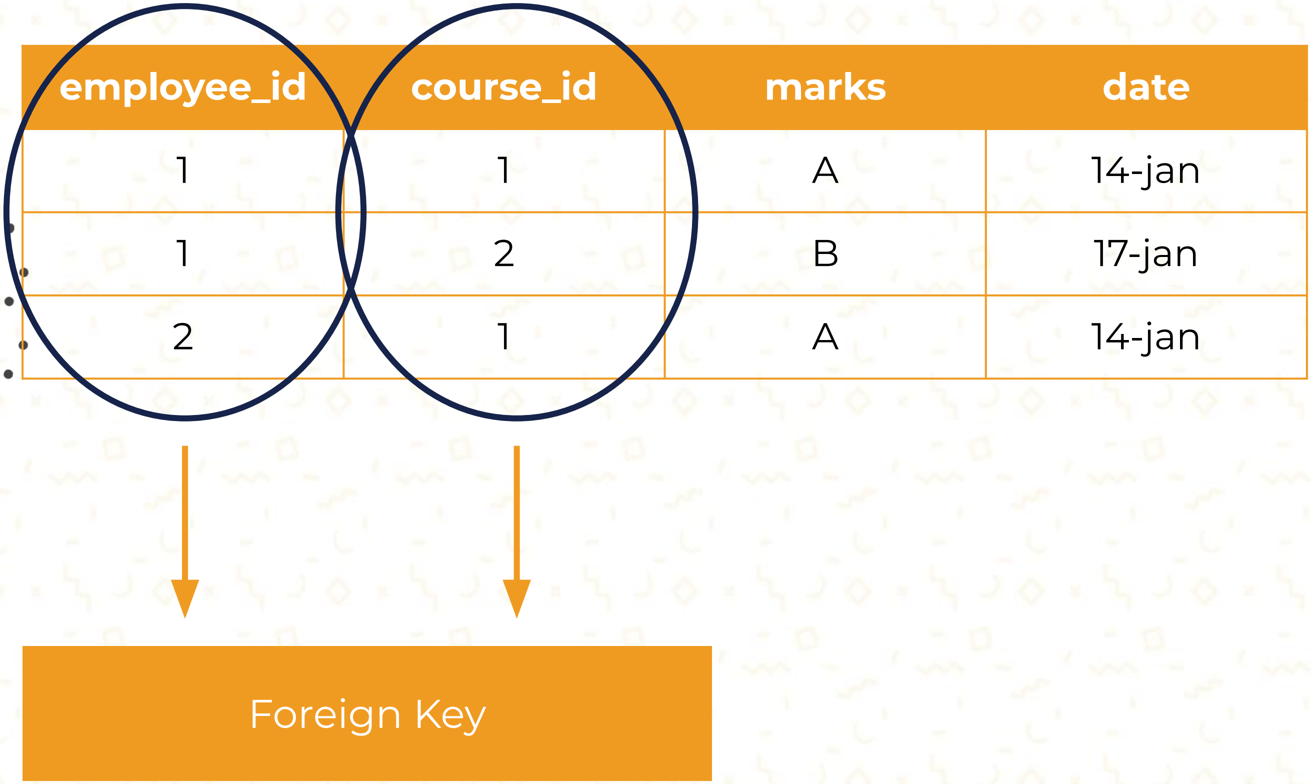
Possible Super Key:

- **employee_id**
- **phone**
- **{student_id, name}**
- **{name, phone}**

Possible Candidate Key:

- **employee_id**
- **phone**

Key



employee_id	course_id	marks	date
1	1	A	14-jan
1	2	B	17-jan
2	1	A	14-jan

Foreign Key

Composite Key:

- **employee_id >< course_id**

Foreign Key:

- **employee_id**
- **course_id**

Database Normalization

Normalization is the process of minimizing redundancy from a relation or set of relations. Redundancy in relation may cause insertion, deletion and updation anomalies. So, it helps to minimize the redundancy in relations. Normal forms are used to eliminate or reduce redundancy in database tables.

Consider this example. You want to update a customer name in a table. Your table consists of a lot of same customer name, may be a same person, may be not. Database normalization could help this problem.

Database Normalization

First Normal Form (1NF)

Rules:

1. Single valued attributes: Each column must not contain multiple values.
2. Attribute domains should not change: The values stored in each column must be of the same kind of type.
3. Unique name for each attribute/column: Each column in a table should have a unique name to avoid confusion.
4. Order doesn’t matters.

Database Normalization

First Normal Form (1NF)

invoice_id	date	item_id	item_name	price	qty
1	17 Jan	a	ABC	10	3
		b	DEF	20	2
2	17 Jan	a	ABC	10	5

invoice_id	date	item_id	item_name	price	qty
1	17 Jan	a	ABC	10	3
1	17 Jan	b	DEF	20	2
2	17 Jan	a	ABC	10	5

Database Normalization

Second Normal Form (2NF)

Rules:

1. The table should be in the 1NF already.
2. There should be no partial dependency: No proper subset of candidate key determines non-prime attribute.

Database Normalization

First Normal Form (2NF)

invoice_id	date	item_id	item_name	price	qty
1	17 Jan	a	ABC	10	3
1	17 Jan	b	DEF	20	2
2	17 Jan	a	ABC	10	5

invoice_id	item_name	price
a	ABC	10
b	DEF	20

transaction_id	date	qty
1	17 Jan	3
1	17 Jan	2
2	17 Jan	5

Database Normalization

Third Normal Form (3NF)

Rules:

1. The table should be in the 2NF already.
2. There should be no transitive dependency: No non-prime attribute depends on the other non-prime attribute.

Database Normalization

First Normal Form (3NF)

invoice_id	item_name	price
a	ABC	10
b	DEF	20

transaction_id	date	qty
1	17 Jan	3
1	17 Jan	2
2	17 Jan	5

invoice_id	item_name	price
a	ABC	10
b	DEF	20

transaction_id	date
1	17 Jan
1	17 Jan
2	17 Jan

invoice_id	item_id	price	qty
1	a	10	3
1	b	20	2
2	a	10	5

DigitalSkola
Uncover The World Of Digital Skills With Us

DATABASE - Data Models (Review)

Relationship Cardinality

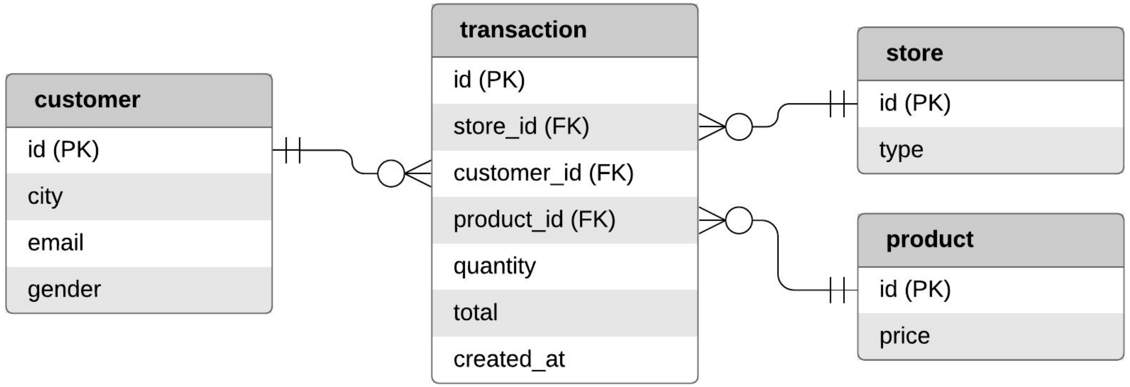
- one to one (1:1)
- one to many (1:M)
- many to many (M:N)

Participant Constraint

- Mandatory
  - At least there's one entity that associated with entity A
- Optional
  - Allowed to have none entity that associated with entity A

Entity types

- Strong: Can stand alone
- Weak: Need others
- Associative: Created by other entities



customer
id (PK)
city
email
gender

transaction
id (PK)
store_id (FK)
customer_id (FK)
product_id (FK)
quantity
total
created_at

store
id (PK)
type

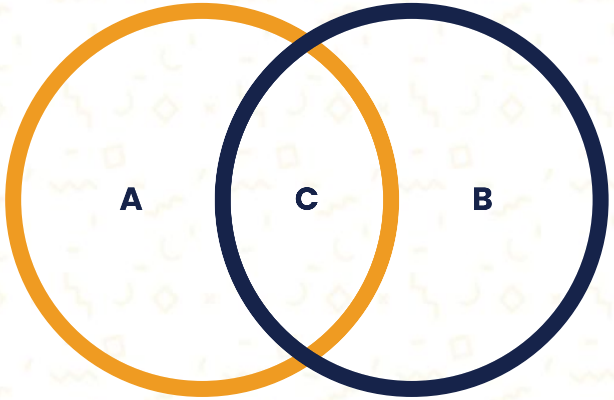
product
id (PK)
price

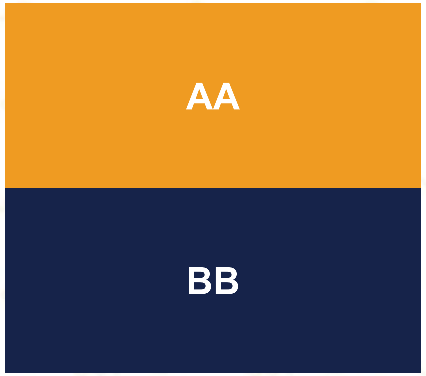
Join and Merge

SQL gives us a capability to do query from more than one table. We could either join them to add the column or merge them to add the row

Join Function:

- LEFT JOIN (A + C)
- RIGHT JOIN (B + C)
- INNER JOIN (C Only)
- FULL OUTER JOIN (A + C + B)



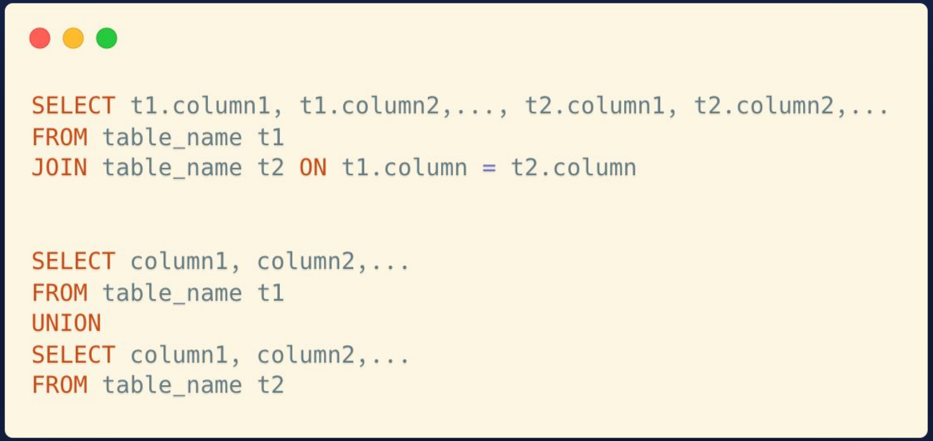


Merge Function:

- UNION (only distinct values)
- UNION ALL (allows duplication)

Join and Merge Function

Used to extract result from more than 1 table. It can be equally similar (merge) or has relation with (join)



```
SELECT t1.column1, t1.column2,..., t2.column1, t2.column2,...
FROM table_name t1
JOIN table_name t2 ON t1.column = t2.column


SELECT column1, column2,...
FROM table_name t1
UNION
SELECT column1, column2,...
FROM table_name t2
```

Tips:

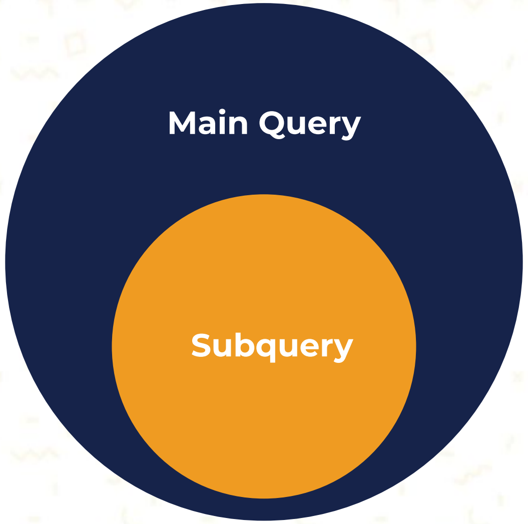
1. Use it accordingly. More tables can cost more to the query performance.
2. Use aliases that reflects the table name eg: customer_transaction_history aliased as cth

Subquery

The next level of SQL queries is to put query inside query. The position of the subquery is depend on the result that we want to retrieve.

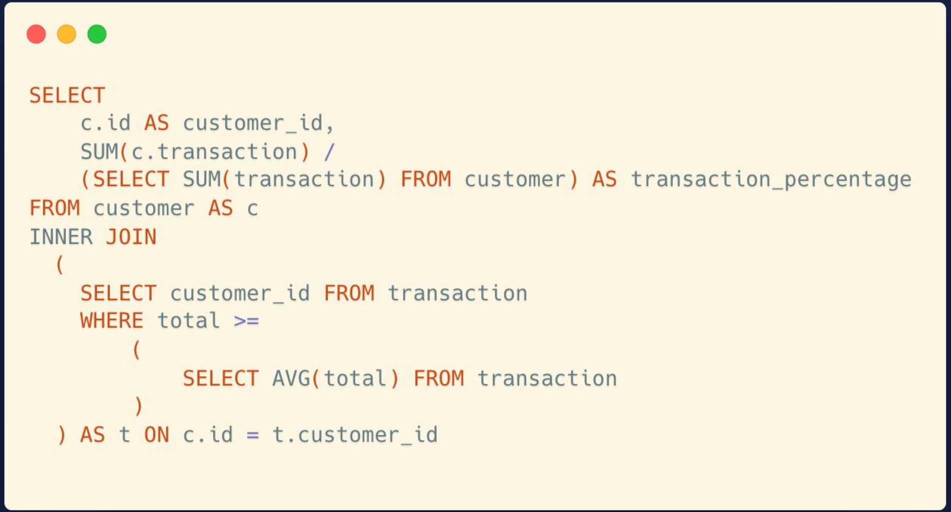
Subquery can be positioned in:

- Inside SELECT
- Inside FROM
- Inside WHERE
- Inside JOIN
- Inside subquery



Subquery

Used to extract result from more than 1 table. It can be quite similar to join function sometimes.



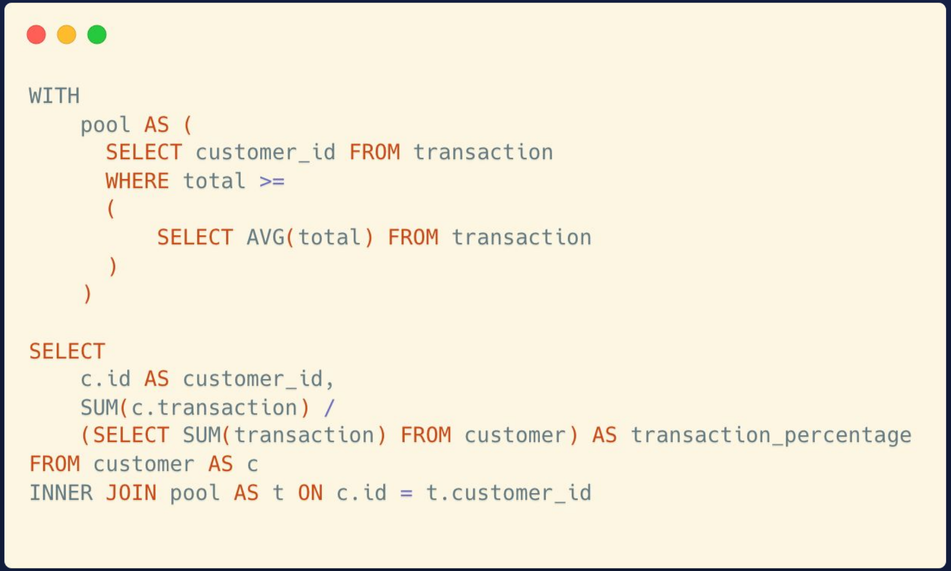
```
SELECT
    c.id AS customer_id,
    SUM(c.transaction) /
    (SELECT SUM(transaction) FROM customer) AS transaction_percentage
FROM customer AS c
INNER JOIN
  (
    SELECT customer_id FROM transaction
    WHERE total >=
        (
            SELECT AVG(total) FROM transaction
        )
  ) AS t ON c.id = t.customer_id
```

Tips:

1. Use it accordingly. More subqueries impact to the query performance significantly.
2. Use aliases if we put subquery under FROM or JOIN function
3. Use aliases that reflects the table name eg: customer_transaction_history aliased as cth

Subquery using WITH

Same purpose with basic subquery but WITH would create all the subqueries into tables in the beginning.



```
WITH
    pool AS (
      SELECT customer_id FROM transaction
      WHERE total >=
      (
          SELECT AVG(total) FROM transaction
      )
    )

SELECT
    c.id AS customer_id,
    SUM(c.transaction) /
    (SELECT SUM(transaction) FROM customer) AS transaction_percentage
FROM customer AS c
INNER JOIN pool AS t ON c.id = t.customer_id
```

Tips:

1. Use it accordingly. More subqueries impact to the query performance significantly.
2. Use aliases if we put subquery under FROM or JOIN function

DigitalSkola
Uncover The World Of Digital Skills With Us

Bonus

Effective Query

Technical Point of View

To solve technical challenge, we must make our query effective.

Some tips:

1. Long query doesn't make it looks cool and complex
2. Be aware with unnecessary function
3. Pick needed columns only: don’t using (“*”) inside every SELECT
4. Order the data if it’s necessary only
5. Reduce the usage of CASE WHEN function if possible
6. Don’t depends on subquery
7. Always put LIMIT if you only want to check the table

Effective Query

Business Point of View

Converting questions from business person might be a challenge as they don’t know how our data structure is. Top down might be the best approach!

Some tips:

1. Extract the goal of the question
2. List down all columns needed to be extracted based on the goal
3. Break down all the sources for columns needed
4. Write SQL script per source to get each columns needed
5. Compile everything into one script

DigitalSkola

Uncover The World Of Digital Skills With Us

Thank YOU