Data Modelling

Sancheeta Kaushal | Ex-EM Blinkit

Few Disclaimers

1. Integrating DB and Code in the real world.

2. Interaction with Cameras on.

3. Nobody knows everything. Let’s learn together.

4. Support me by telling me when you haven’t understood the concepts.

Agenda

- Introduction to Joins
- Breakout room
- ORM’s
- Setting up ORM
- Let’s write CRUD API’s
- Assignment

Airtribe

Main takeaway: You will learn the concepts of joins, aggregations and ORM's.

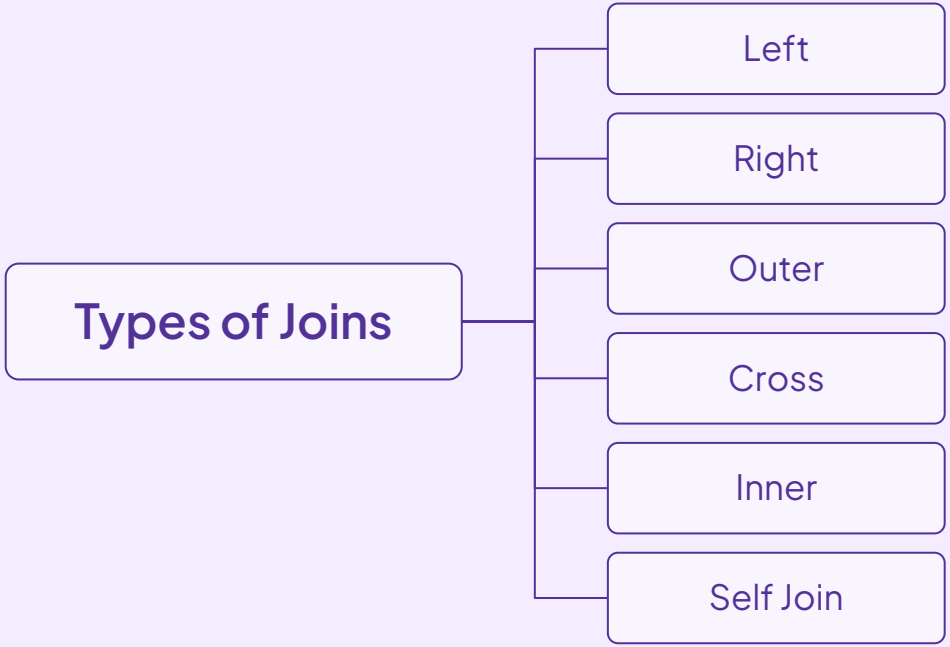
Learning outcome: You should be able to integrate the database with your code and write some API's to do CRUD operations.

Normalisation

- Process of organising data in a database.
- Grouping of similar values as one group.
- Eliminate redundancy and inconsistent dependency.
- We still need to aggregate data to solve for real life scenarios.

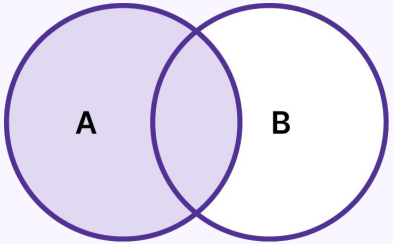
Joins

Way to combine data from two or more tables based on the related columns.

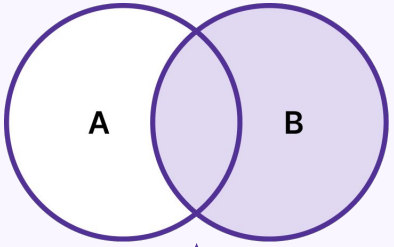


Airtribe

LEFT & RIGHT JOIN – https://bit.ly/3MssiSc




```
SELECT A.column, B.column
FROM TableA A
LEFT JOIN TableB B
ON A.Key = B.Key
```




```
SELECT A.column, B.column
FROM TableA A
RIGHT JOIN TableB B
ON A Key = B.Key
```

Airtribe

Example – Left Join

You have customers and orders table. You want to list down all the customers and their corresponding order information whether they have placed any order yet or not.

SQL Query Syntax

```
SELECT A.column, B.column FROM TableA A
LEFT JOIN TableB B
ON A.Key = B.Key;
```

SQL Query Syntax

```
SELECT customers.customer_name, orders.order_id, orders.order_date,
orders.total_amount FROM customers LEFT JOIN orders
ON customers.customer_id = orders.customer_id;
```

Airtribe

Example – Right Join

You have customers and orders table. You want to list down all the orders and and their corresponding customer information if available.

SQL Query Syntax

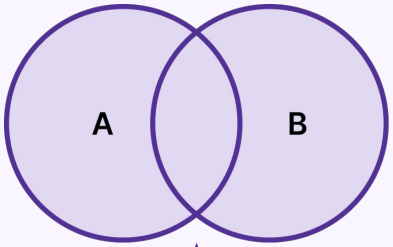
```
SELECT A.column, B.column FROM TableA A
RIGHT JOIN TableB B
ON A.Key = B.Key;
```

SQL Query Syntax

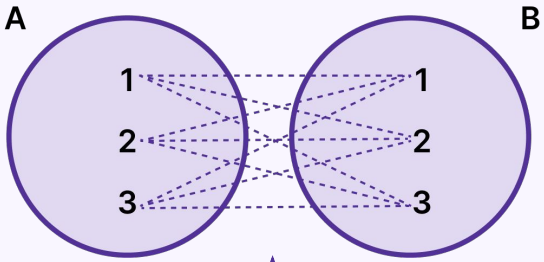
```
SELECT customers.customer_name, orders.order_id, orders.order_date,
orders.total_amount FROM customers RIGHT JOIN orders
ON customers.customer_id = orders.customer_id;
```

Airtribe

OUTER & CROSS JOIN – https://bit.ly/3MssiSc




```
SELECT A.column, B.column
FROM TableA A
FULL OUTER JOIN TableB B
ON A.Key =B.Key
```




```
SELECT A.column, B.column
FROM TableA A
CROSS JOIN TableB B
ON A.Key =B.Key
```

Airtribe

Example – Outer Join

You have customers and orders table. You want to list down all the orders for a all the customers.

SQL Query Syntax

```
SELECT A.column, B.column FROM TableA A
FULL OUTER JOIN TableB B
ON A.Key = B.Key;
```

SQL Query Syntax

```
SELECT customers.customer_name, orders.order_id, orders.order_date,
orders.total_amount FROM customers LEFT JOIN orders
ON customers.customer_id = orders.customer_id
UNION ALL
SELECT customers.customer_name, orders.order_id, orders.order_date,
orders.total_amount FROM customers RIGHT JOIN orders
ON customers.customer_id = orders.customer_id;
```

Example – Cross Join

You have customers and orders table. You want to create a cartesian product of rows in both tables.

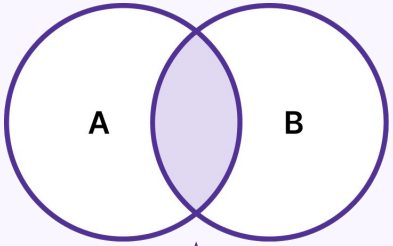
SQL Query Syntax

```
SELECT A.column, B.column FROM TableA A
CROSS JOIN TableB B;
```

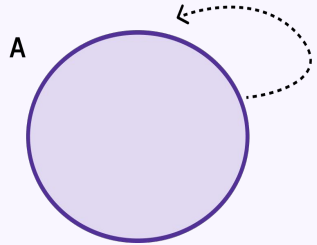
SQL Query Syntax

```
SELECT customers.customer_name, orders.order_id, orders.order_date,
orders.total_amount FROM customers CROSS JOIN orders;
```

INNER & SELF JOIN – https://bit.ly/3MssiSc




```
SELECT A.column, B.column
FROM TableA A
INNER JOIN JOIN TableB B
ON A.Key =B.Key
```




```
SELECT A1.column, A2.column
FROM TableA A1
LEFT JOIN TableA A2
ON A1.Key =A2.Key
```

Airtribe

Example – Inner Join

You have customers and orders table. You want to list down the orders which have a corresponding customer in both the tables.

SQL Query Syntax

```
SELECT A.column, B.column FROM TableA A
INNER JOIN TableB B
ON A.Key = B.Key;
```

SQL Query Syntax

```
SELECT customers.customer_name, orders.order_id, orders.order_date,
orders.total_amount FROM customers INNER JOIN orders
ON customers.customer_id = orders.customer_id;
```

Example – Self Join

You have customers table. You want to find all the customers who belong to the same city.

SQL Query Syntax

```
SELECT A1.column, A2.column FROM TableA1 A1
LEFT JOIN TableA2 A2
ON A1.Key = A2.Key;
```

SQL Query Syntax

```
SELECT c1.customer_id, c1.customer_name, c2.customer_id, c2.customer_name
FROM customers c1
JOIN customers c2 ON c1.city = c2.city AND c1.customer_id <> c2.customer_id
ORDER BY c1.customer_id, c2.customer_id;
```

Data Aggregation and Ordering

- Group by and Having clause
- Aggregate functions like COUNT, SUM, AVG, MAX, or MIN
- Having vs Where - Full table scan
- Order by - Ascending or descending

Example – Group by

You have orders table and you want to find out how much revenue is generated by each customer.

SQL Query Syntax

```
SELECT Column1, aggregate_function(Column2)
FROM TableA
GROUP BY Column1;
```

SQL Query Syntax

```
SELECT customer_id, SUM(total_amount) as revenue
FROM orders
GROUP BY customer_id;
```

Example – Having

You have orders table and you want to find out how much revenue is generated by sales of each product and select only the products whose revenue is greater than 1050.

SQL Query Syntax

```
SELECT Column1, aggregate_function(Column2) as Column
FROM TableA
GROUP BY Column1 HAVING Column > X;
```

SQL Query Syntax

```
SELECT customer_id, SUM(total_amount) as revenue
FROM orders
GROUP BY customer_id HAVING revenue > 1050;
```

Example – Order by

Order the output of previous example in the descending order of revenue.

SQL Query Syntax

```
SELECT *
FROM Table
ORDER BY Column ASC/DESC;
```

SQL Query Syntax

```
SELECT customer_id, SUM(total_amount) as revenue
FROM orders
GROUP BY customer_id HAVING revenue > 1050 order by revenue desc;
```

Example – Grouping and Ordering

Given the data for customers and orders, answer the following question

- How many orders were placed by each city, ordered in ascending order of the city name?

SQL Query Syntax

```
SELECT c.city, COUNT(o.order_id) AS order_count
FROM customers c
LEFT JOIN orders o ON c.customer_id = o.customer_id
GROUP BY c.city
ORDER BY c.city ASC;
```

Problem Solving – P1 – https://bit.ly/3MssiSc

What is the average total amount spent by customers from each city, for orders placed after 3rd May 2022, ordered in descending order of the average amount?

customer_id	customer_name	city	country
1	John Smith	New York	USA
2	Jane Doe	San Francisco	USA
3	Bob Johnson	London	UK
4	Alice Brown	Paris	France
5	Tom Lee	Tokyo	Japan
6	Sancheeta	Bengaluru	India
8	Dhaval	Bengaluru	India

order_id	customer_id	product_id	order_date	total_amount
1	1	1	2022-05-01	500
2	1	2	2022-05-05	1,000
3	2	2	2022-05-03	750
4	2	3	2022-05-07	300
5	3	1	2022-05-02	1,250
6	3	3	2022-05-08	900
7	4	1	2022-05-06	800
8	4	2	2022-05-09	650
9	5	1	2022-05-04	1,100
10	5	3	2022-05-10	950

10 - Min Live Exercise

Airtribe

Problem Solving

What is the average total amount spent by customers from each city, for orders placed after 3rd May 2022, ordered in descending order of the average amount?

SQL Query Syntax

```
SELECT c.city, AVG(o.total_amount) AS average_amount
FROM customers c
JOIN orders o ON c.customer_id = o.customer_id
WHERE o.order_date >= '2022-05-03'
GROUP BY c.city
ORDER BY average_amount DESC;
```

Problem Solving – P2

Write a query that retrieves the customer name, city, product category, and the total number of orders made for each product category. Group the result by the customer name and product category ?

customer_id	customer_name	city	country
1	John Smith	New York	USA
2	Jane Doe	San Francisco	USA
3	Bob Johnson	London	UK
4	Alice Brown	Paris	France
5	Tom Lee	Tokyo	Japan
6	Sancheeta	Bengaluru	India
8	Dhaval	Bengaluru	India

order_id	customer_id	product_id	order_date	total_amount
1	1	1	2022-05-01	500
2	1	2	2022-05-05	1,000
3	2	2	2022-05-03	750
4	2	3	2022-05-07	300
5	3	1	2022-05-02	1,250
6	3	3	2022-05-08	900
7	4	1	2022-05-06	800
8	4	2	2022-05-09	650
9	5	1	2022-05-04	1,100
10	5	3	2022-05-10	950

product_id	product_name	category
1	Phone	Electronics
2	Laptop	Electronics
3	Shirt	Clothing

10 - Min Live Exercise

Airtribe

Problem Solving

Write a query that retrieves the customer name, city, product category, and the total number of orders made for each product category. Group the result by the customer name and product category ?

SQL Query Syntax

```
SELECT
customers.customer_name, customers.city, products.category,
COUNT(order_details.order_id) AS total_orders FROM customers JOIN
order_details ON customers.customer_id = order_details.customer_id JOIN
products ON order_details.product_id = products.product_id
GROUP BY customers.customer_name, customers.city, products.category;
```

Airtribe

How do applications interact with database ?

ORM - Object Relational Mapper

Object (Apps) → Relations (Tables)

Done via models

Setting up ORM in Codebase

- You have already installed the mysql server (brew install) and mysql client (DBeaver).
- Now you need to install mysql NPM Library and Sequalize ORM in package.json as dependencies.
- Next we create a config to write code to connect with MySQL.
- Once connection is tested, let's create the product model for our Instamart use case.
- Run the migration to create the table in the database. You can learn how to run migrations on production at https://sequelize.org/docs/v6/other-topics/migrations/

Creation & persistence to database

build, create, save, bulkCreate

```
const User = sequelize.define('User', {
  name: Sequelize.STRING,
  age: Sequelize.INTEGER,
});

// Build a new user instance
const newUser = User.build({ name: 'John', age: 25 });

// Save the new user to the database
await newUser.save();

console.log(newUser);

// Create a new user directly in the database
const createdUser = await User.create({ name: 'Jane', age: 30 });

console.log(createdUser);

// Bulk create multiple users in a single operation
const usersData = [
  { name: 'Alice', age: 28 },
  { name: 'Bob', age: 32 },
];

const createdUsers = await User.bulkCreate(usersData);
```

Airtribe

Update

```
User.update(
  { age: 30 }, // Updated values
  {
    where: { id: 1 }, // Condition for updating specific records
  }
)
  .then((result) => {
    console.log(result); // Number of rows affected
  })
  .catch((error) => {
    console.error(error);
  });
```

Airtribe

Delete & Reload

```
const User = sequelize.define('User', {
  name: Sequelize.STRING,
  age: Sequelize.INTEGER,
});

// Find a user by ID
const user = await User.findByPk(1);

// Reload the user from the database
await user.reload();

console.log(user);

// Update the user's attributes
user.name = 'John Doe';
user.age = 30;

// Save the updated user to the database
await user.save();

console.log(user);

// Destroy the user
await user.destroy();

console.log('User deleted');
```

Airtribe

Find in Sequelize

findAll, findOne, findByPk, findOrCreate

```
const User = sequelize.define('User', {
  name: Sequelize.STRING,
  age: Sequelize.INTEGER,
});

// findAll: Retrieve all users
const allUsers = await User.findAll();
console.log(allUsers);

// findOne: Retrieve a single user
const user = await User.findOne({ where: { id: 1 } });
console.log(user);

// findByPk: Retrieve a user by primary key
const userById = await User.findByPk(1);
console.log(userById);

// findOrCreate: Find or create a user
const [foundUser, created] = await User.findOrCreate({
  where: { name: 'John' },
  defaults: { age: 25 },
});
```

Airtribe

Associations

HasOne, BelongsTo, HasMany, BelongsToMany

```
const User = sequelize.define('User', {
  name: Sequelize.STRING,
});

const UserProfile = sequelize.define('UserProfile', {
  bio: Sequelize.TEXT,
});

User.hasOne(UserProfile);
UserProfile.belongsTo(User);
```

```
const User = sequelize.define('User', {
  name: Sequelize.STRING,
});

const Post = sequelize.define('Post', {
  title: Sequelize.STRING,
  content: Sequelize.TEXT,
});

User.hasMany(Post);
Post.belongsTo(User);
```

```
const User = sequelize.define('User', {
  name: Sequelize.STRING,
});

const Project = sequelize.define('Project', {
  title: Sequelize.STRING,
});

const UserProject = sequelize.define('UserProject', {
  role: Sequelize.STRING,
});

User.belongsToMany(Project, { through: UserProject });
Project.belongsToMany(User, { through: UserProject });
```

Airtribe

Filtering & ordering parameters

Where, order, limit, offset, include (easy loading concept)

```
const User = sequelize.define('User', {
  name: Sequelize.STRING,
  age: Sequelize.INTEGER,
});

const Post = sequelize.define('Post', {
  title: Sequelize.STRING,
  content: Sequelize.TEXT,
});

User.hasMany(Post);
Post.belongsTo(User);

// Retrieve users with age greater than 25, order by name in descending orde
// limit the results to 10, and include associated posts
const users = await User.findAll({
  where: { age: { [Op.gt]: 25 } },
  order: [['name', 'DESC']],
  limit: 10,
  include: [{ model: Post }],
});
```

Airtribe

Grouping Parameters

```
const User = sequelize.define('User', {
  name: Sequelize.STRING,
  age: Sequelize.INTEGER,
  city: Sequelize.STRING,
});

// Group users by city and retrieve the count of users in each city
const groupedUsers = await User.findAll({
  attributes: ['city', [sequelize.fn('COUNT', sequelize.col('id')), 'userCount']],
  group: ['city'],
});
```

Thank You!

Airtribe