there was points but you removed

Hibernate Notes (Complete Recall Sheet)

1. What is Hibernate?

- ORM framework for Java.
- Maps Java objects ↔ DB tables.
- Eliminates most SQL boilerplate.

2. Core Components

- Configuration → hibernate.cfg.xml (DB details, dialect).
- SessionFactory → Heavy object, created once.
- Session → Light, per unit of work.
- Transaction → Group of operations (commit/rollback).

3. Common Configs (hibernate.cfg.xml)

```
<property name="hibernate.dialect">org.hibernate.dialect.PostgreSQLDialect</property>
<property name="hibernate.hbm2ddl.auto">update</property> <!-- dev only -->
<property name="hibernate.show_sql">true</property>
<property name="hibernate.format_sql">true</property>
```

4. Entity Basics

```
@Entity
@Table(name = "users")
public class User {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    @Column(nullable = false, unique = true)
    private String email;

    private String name;
}
```

Annotations:

- @Entity → Marks class as persistent.

- @Table → Custom table name.
- @Id → Primary key.
- @GeneratedValue → Auto ID strategy.
- @Column → Column details.

5. Relationships

- One-to-One → @OneToOne + @JoinColumn
- One-to-Many / Many-to-One → @OneToMany(mappedBy="..."), @ManyToOne
- Many-to-Many → @ManyToMany + @JoinTable
  Default fetch:
- @OneToOne, @ManyToOne → EAGER
- @OneToMany, @ManyToMany → LAZY

6. Queries
HQL (Hibernate Query Language)

```
session.createQuery("from User where name=:n", User.class)
       .setParameter("n", "John")
       .list();
```

Criteria API

```
CriteriaBuilder cb = session.getCriteriaBuilder();
CriteriaQuery<User> cq = cb.createQuery(User.class);
Root<User> root = cq.from(User.class);
cq.select(root).where(cb.equal(root.get("name"), "John"));
```

Native SQL

```
session.createNativeQuery("SELECT * FROM users", User.class).list();
```

7. Transactions

```
Transaction tx = session.beginTransaction();
try {
    // operations
    tx.commit();
} catch(Exception e) {
    tx.rollback();
}
```

- Always use transactions.
- Prevents partial updates.

8. Caching

- First-Level (default) → Per Session.
- Second-Level → Across sessions, needs provider.

```
@Entity
@Cacheable
@org.hibernate.annotations.Cache(usage = CacheConcurrencyStrategy.READ_WRITE)
public class User {}
```

9. Inheritance Mapping

- SINGLE_TABLE (one table, discriminator column).
- JOINED (normalized, slower).
- TABLE_PER_CLASS (separate tables, least common).
  Example:

```
@Inheritance(strategy = InheritanceType.SINGLE_TABLE)
@DiscriminatorColumn(name="type")
```

10. Common Pitfalls
✕ N+1 queries

```
// BAD
for(User u : session.createQuery("from User").list()) {
    u.getOrders().size();
}
```

✓ **Fix with JOIN FETCH**

```
session.createQuery("select u from User u join fetch u.orders", User.class).list();
```

- ✕ Overusing eager fetch → performance crash.
- ✕ Using hbm2ddl.auto=update in prod.

11. Best Practices

- Use lazy loading by default.
- Wrap all ops in transactions.
- Use DTOs for APIs, don't expose entities.
- Use batch fetching for collections.
- Monitor queries with SQL logging.
- Use Flyway/Liquibase for schema migrations.

12. Integration

- Works standalone or with Spring Boot (JPA).
  With Spring Data JPA:

```
public interface UserRepository extends JpaRepository<User, Long> {
    List<User> findByName(String name);
}
```

✅ This single sheet should help you recall all important Hibernate concepts quickly.