

Module 2

Optimizing Application Design

Created by [Teerachai Laothong](#)

Agenda

- Connection Management
- Parsing SQL Statements
- Sharing Reusable Code
- Schema Denormalization
- Avoiding Dynamic SQL

Connection Management

- Use Connection Pooling
- Hold database connection as short as possible

Dedicated Server versus Shared Server

There are two types of configurations used to connect to an Oracle database: dedicated server and shared server, called Multi-Threaded Server (MTS) in previous Oracle database releases.

Dedicated Server

In dedicated server mode, each client connected to the database will have a separate server process that executes the requests.

The dedicated server configuration is configured by default in most environments.

Shared Server

In a shared server configuration, when a client initiates a connection to the database, the listener process chooses a dispatcher process configured for the database and the dispatcher process passes the request to the least loaded server process, which is already configured for the database, to execute the client request.

Web applications 1/2

Web applications pool many users connecting at the same time, and this is the first thing to consider when designing the connection management of our web application.

We need to *use connection pooling* when connecting to the database in our web application.

Web applications 2/2

A typical pattern for a web page design:

1. Connect to the database.
2. Query the data and store for use later in code.
3. Close the connection.

Batch processing

For batch processing, a dedicated server connection is a must, because they often use large queries and updates, which may last for minutes, if not hours on large systems.

Example

Java Code Sample:

```
public static void main(String[] args) throws SQLException {
    long startTime = System.currentTimeMillis();
    singleConnection();
    long stopTimeSingle = System.currentTimeMillis();
    multipleConnection();
    long stopTimeMulti = System.currentTimeMillis();
    System.out.println(String.format(
        "Execution with single connection %dms.\nExecution with multi
        (stopTimeSingle - startTime),
        (stopTimeMulti - stopTimeSingle)));
}
```

singleConnection

```
public static void singleConnection() throws SQLException {
    Connection conn = null;
    try {
        Class.forName(driver);
        conn = DriverManager.getConnection(connectionString, user, pass);
    } catch (Exception e) {
        System.out.println(String.format("Error %s", e.getLocalizedMessage()));
        System.exit(1);
    }
    try {
        for (int j = 0; j < iterations; ++j) {
            Statement query = conn.createStatement();
            ResultSet result = query.executeQuery("select first_name,last_name from employees");
            while (result.next()) {
                String name = result.getString("first_name") + " " + result.getString("last_name");
                System.out.println(name);
            }
            query.close();
        }
    }
}
```

multipleConnection

```
public static void multipleConnection() throws SQLException {
    Connection conn = null;
    for (int j = 0; j < iterations; ++j) {
        try {
            Class.forName(driver);
            conn = DriverManager.getConnection(connectionString, user, pa
        } catch (Exception e) {
            System.out.println(String.format("Error %s", e.getLocalizedMessage));
            System.exit(1);
        }
        try {
            Statement query = conn.createStatement();
            ResultSet result = query.executeQuery("select first_name, las
            while (result.next()) {
                String name = result.getString("first_name")
                    + " " + result.getString("last_name");
                System.out.println(name);
            }
        }
    }
}
```

Result

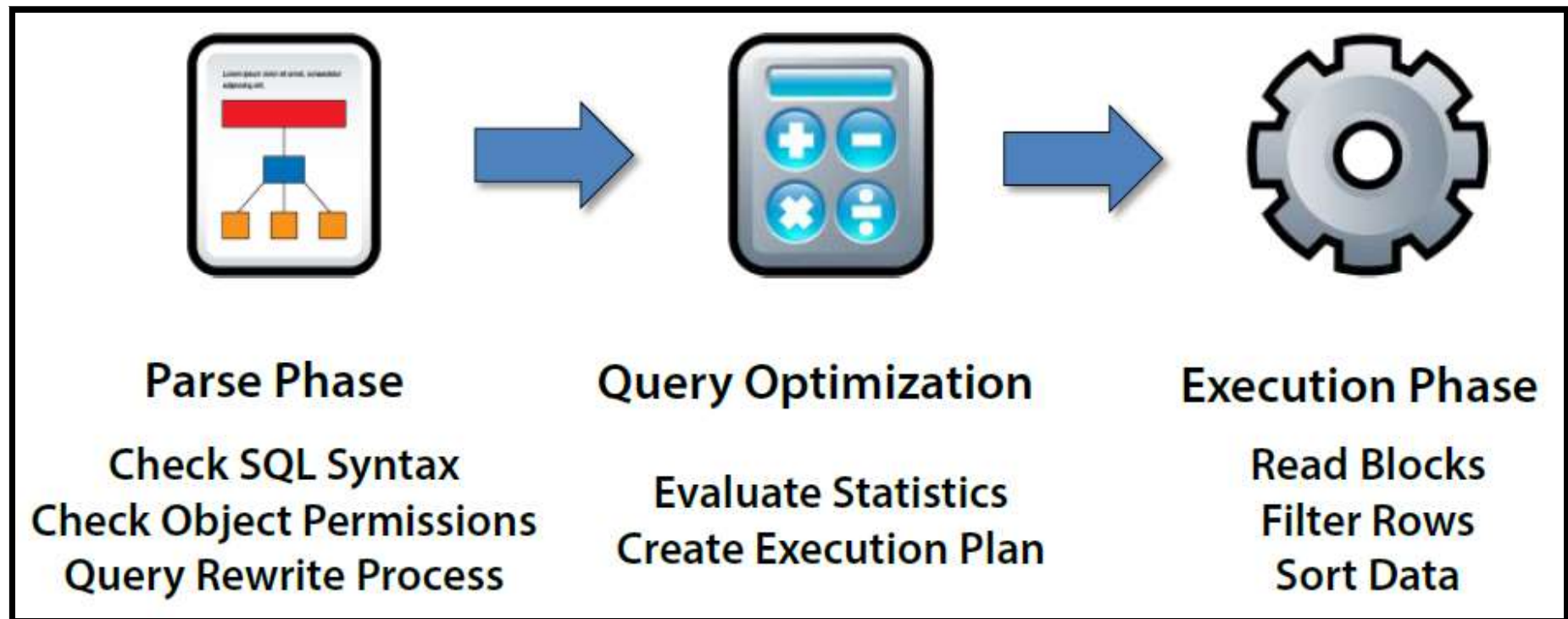
The output will be as follows:

```
...  
Matthew Weiss  
Jennifer Whalen  
Eleni Zlotkey  
Execution with single connection 2863ms.  
Execution with multiple connections 4176ms.
```

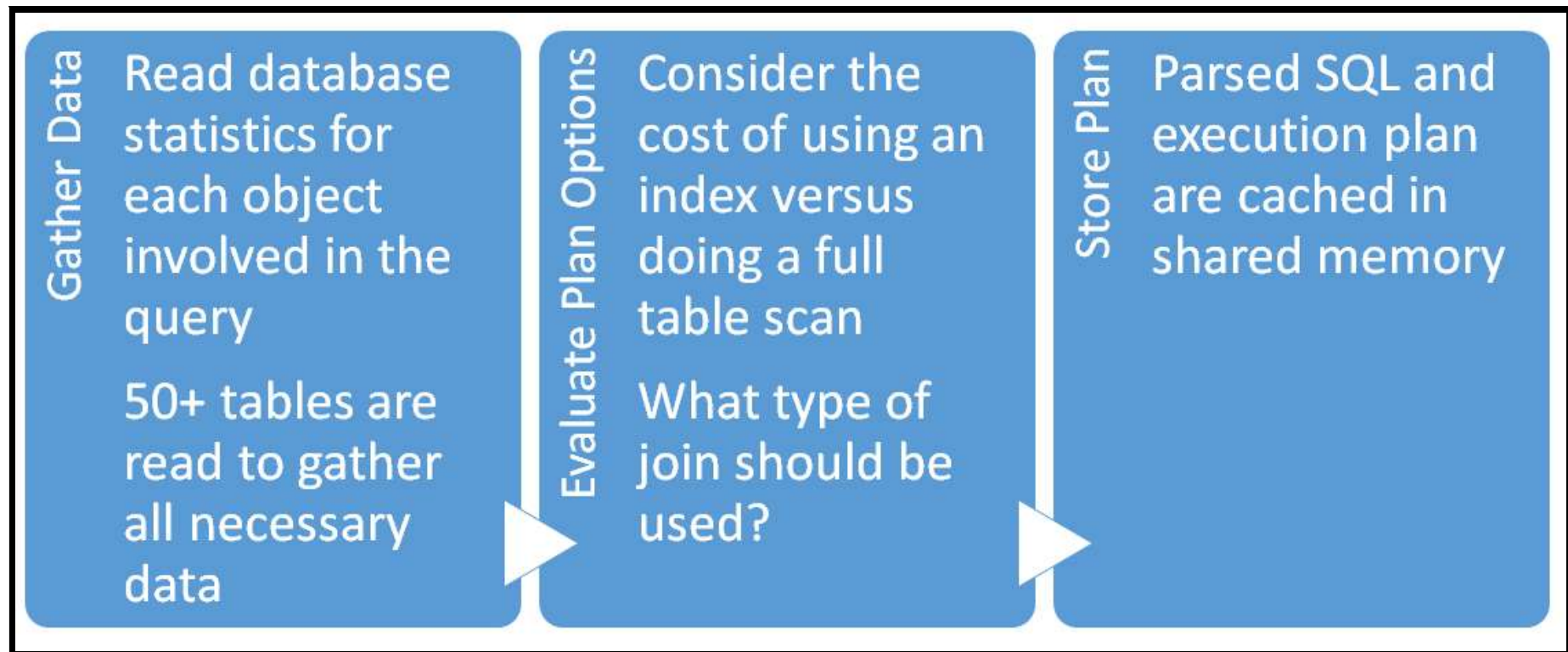
Parsing SQL Statements

- Life of a SQL Statement
- Steps in Creating an Execution Plan
- The Shared SQL Area
- Plan lookup in the Shared SQL Area

Life of a SQL Statement



Steps in Creating an Execution Plan






The Shared SQL Area



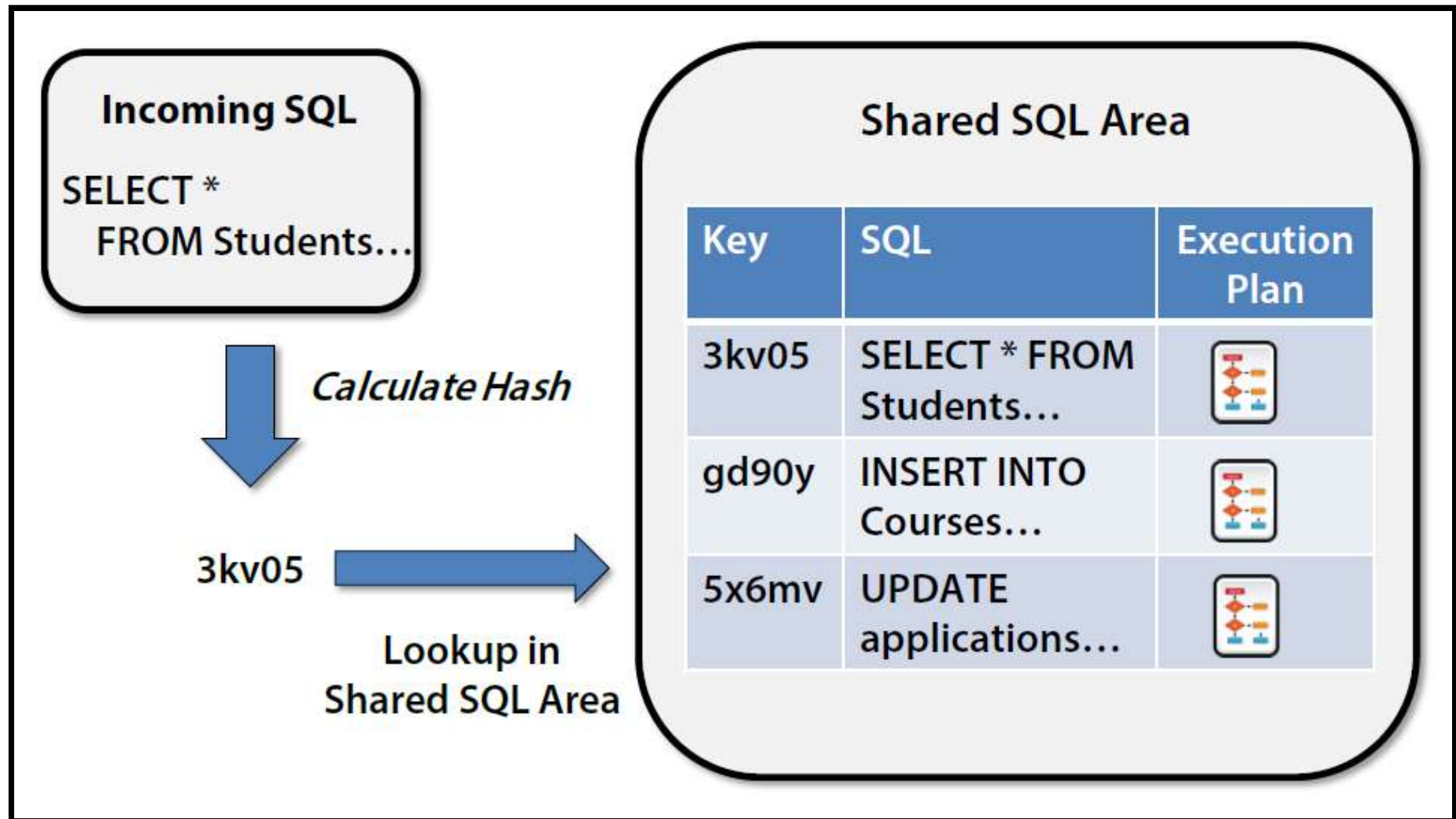
From the Oracle
Optimizer



Shared SQL Area

Key	SQL	Execution Plan
3kv05	SELECT * FROM Students...	
gd90y	INSERT INTO Courses...	
5x6mv	UPDATE applications...	

Plan lookup in the Shared SQL Area



Sharing Reusable Code

- Use Oracle's Stored Procedure
- Or use `prepareStatement`

Reusing statements

The correct way to use SQL statements in our applications:

1. Prepare the statement.
2. Execute the statement many times (using bind variables).
3. Close the statement.

Soft Parses versus Hard Parses

Soft Parse

- Oracle can reuse existing execution plan already in the library cache
- Consists of a lookup in the Shared SQL Area
- Preferable over a hard parse

Hard Parse

- Oracle generates a new execution plan
- All statements must be hard parsed on their first execution or after a period of time such that they are no longer in the Shared SQL Area

PL/SQL and parsing

- In a PL/SQL procedure, we don't need to explicitly prepare a statement before executing it, because the DML statements inside our procedures are automatically parsed once per session, and not once per execution.
- If our application is written in Java, we have to `prepareStatement` in order to execute the statement with one preparation.

Example

Java Code:

```
public static void main(String[] args) throws SQLException {
    Connection conn = null;
    try {
        Class.forName(driver);
        conn = DriverManager.getConnection(connectionString, user, pass);
        long startTime = System.currentTimeMillis();
        singleConnection(conn);
        long stopTimeSingle = System.currentTimeMillis();
        preparedQuery(conn);
        long stopTimePrep = System.currentTimeMillis();
        System.out.println(String.format("Execution without prepared query: %d ms", stopTimeSingle - startTime));
    } catch (Exception e) {
        System.out.println(String.format("Error %s", e.getLocalizedMessage()));
        System.exit(1);
    } finally {
        conn.close();
    }
}
```

singleConnection

```
public static void singleConnection(Connection conn)
    throws SQLException {
    try {
        for (int j = 0; j < iterations; ++j) {
            Statement query = conn.createStatement();
            ResultSet result = query.executeQuery("select first_name, last_name from employees");
            while (result.next()) {
                String name = result.getString("first_name") + " " + result.getString("last_name");
                System.out.println(name);
            }
            query.close();
        }
    } catch (Exception e) {
        System.out.println(String.format("Error %s", e.getLocalizedMessage()));
        System.exit(1);
    }
}
```


preparedQuery

```
public static void preparedQuery(Connection conn) throws SQLException {
    try {
        PreparedStatement ps = conn.prepareStatement("select first_name,
        for (int j = 0; j < iterations; ++j) {
            ResultSet result = ps.executeQuery();
            while (result.next()) {
                String name = result.getString("first_name")
                    + " " + result.getString("last_name");
                System.out.println(name);
            }
        }
        ps.close();
    } catch (Exception e) {
        System.out.println(String.format("Error %s",
            e.getLocalizedMessage()));
        System.exit(1);
    }
}
```

Result

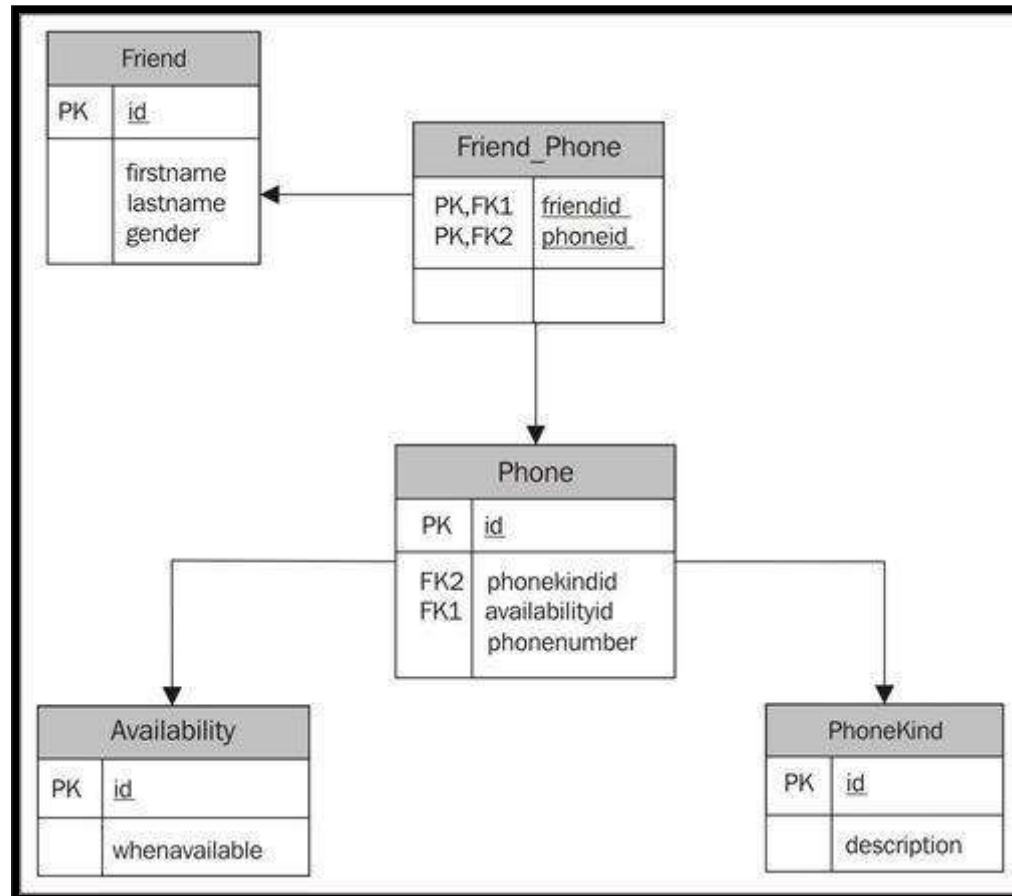
The output will be as follows:

```
...  
Matthew Weiss  
Jennifer Whalen  
Eleni Zlotkey  
Execution without prepared query 15198ms.  
Execution with prepared query 13033ms.
```

Schema Denormalization

- Avoid over-normalization

Example



Query for Over-Normalization

```
SET AUTOTRACE TRACEONLY
SELECT F.FIRSTNAME, F.LASTNAME, PK.DESCRPTION AS PHONEKIND,
       PA.WHENAVAILABLE AS AVAILABILITY, P.PHONENUMBER
FROM FRIEND F
      INNER JOIN FRIEND_PHONE FP ON FP.FRIENDID = F.ID
      INNER JOIN PHONE P ON P.ID = FP.PHONEID
      INNER JOIN PHONEKIND PK ON PK.ID = P.PHONEKINDID
      LEFT OUTER JOIN AVAILABILITY PA ON PA.ID = P.AVAILABILITYID
WHERE F.ID = 29912;
SET AUTOTRACE OFF
```

Query after Denormalization

```
SET AUTOTRACE TRACEONLY
SELECT F.FIRSTNAME, F.LASTNAME, P.PHONEKIND, P.AVAILABILITY,
       P.PHONENUMBER
FROM FRIEND F
      INNER JOIN FRIEND_PHONE FP ON FP.FRIENDID = F.ID
      INNER JOIN PHONE P ON P.ID = FP.PHONEID
WHERE F.ID = 29912;
SET AUTOTRACE OFF
```

Result

- Recursive call will reduced from 1739 to 1078
- Consistent gets will reduced from 405 to 241

Avoiding Dynamic SQL

- If possible never use dynamic SQL
- Dynamic SQL is a powerful feature of the database but it should be used carefully.
- We will investigate this further in Using bind variables in module 4.

When to use Dynamic SQL

Dynamic SQL is the only choice when:

- We want to execute DDL statements in our application.
- We have to code different queries depending on user input, for example, a search form with different search criteria that the user can choose from. This leads to different predicates in the WHERE clause.
- We want to code generic procedures, which can act on any table, for example, a generic print procedure, which shows the content of a table in a certain format.

THE END

- [Source code & documentation](#)
- [Back to Course Outline](#)