

Working With Relational Database Management Systems

Using: Microsoft SQL Server and Oracle 10g

Intro to SQL Server

- Relational Database Management System
 - What is it?
- Database – container for objects that store data and retrieve data in a safe and secure manner
- Tables – where the data is stored, contain one or more columns
- Columns – provides a definition of each single item of information, that builds up to a table definition. Each has it's own data type.
- Rows – also called records, define a single unit of information

Database System Types

- Online Transaction Processing(OLTP)
 - What is it?
 - Updates to database are instant
 - Normalized to 3NF
 - Data updates are kept as short as possible
 - High number of Indexes – faster speed

Database System Types (cont.)

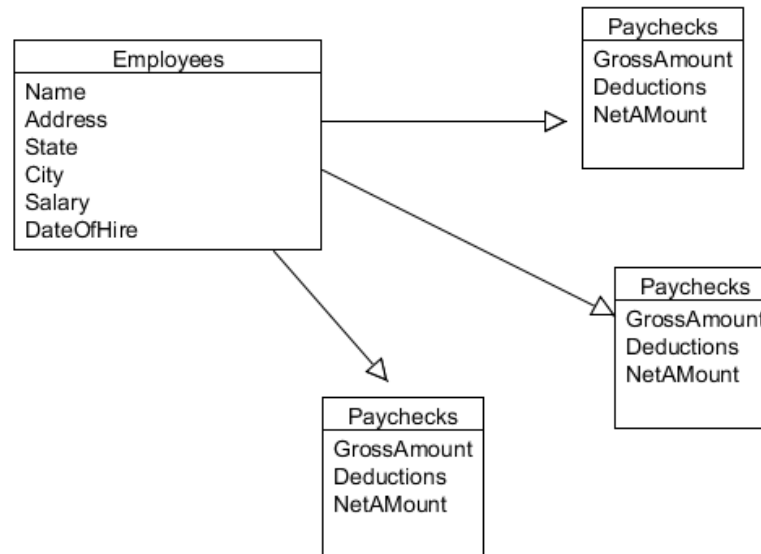
- Online Analytic Processing(OLAP)
 - Static data, infrequent updates
 - No normalization, few indexes
 - Also know as “Data Warehouses”

Database Design

- Important points to think about or do when designing a DB:
 - What kind of data will the database be storing?
 - Gathering info from the business users
 - Organizing info into logical units
 - Employees have common properties such as Name, Address, SSN, DateOfHire, Salary, etc.
 - What kind of relationships exist between units
 - One employee will receive multiple paychecks, which have their own distinct properties.

Database Design(cont.)

What are some other tables we could add?



Relationships

- Now that we have discussed DB design, let's take a look at tables and their relationships with other tables
- A relationship in a RDBMS such as SQL Server is a logical link between two tables.
- Relationships are established through the use of keys
 - Keys are a way identifying a record or row in a table:
 - Primary keys – a column, or columns, which serve as the unique identifiers of a row and can not be null.
 - Foreign key – references primary key in another table. Does not need to be unique

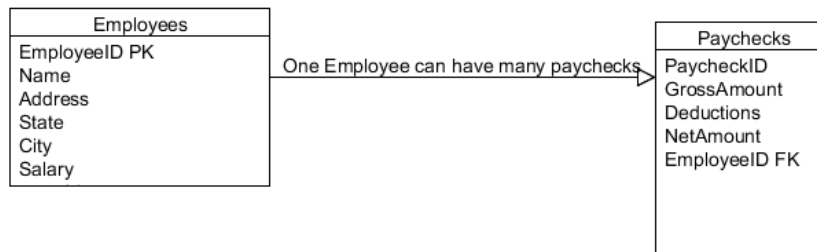
1:n Relationship

- A one to many relationship(1:n) is established by using a PK in a table, and referencing that PK in another table as a FK.
- Think about it: One unique employee(unique), must be a present employee(not null) can receive many paychecks. However, a paycheck can only be sent to one specific, present employee.
- A 1:n relationship is very common in a Database
- Other 1:n relationships:
 - Customer to Orders
 - Dept to employees

1:n Relationship

Employees
contains PK –
unique, not null

Paychecks contains the
EmployeeID FK, does not
need to be unique



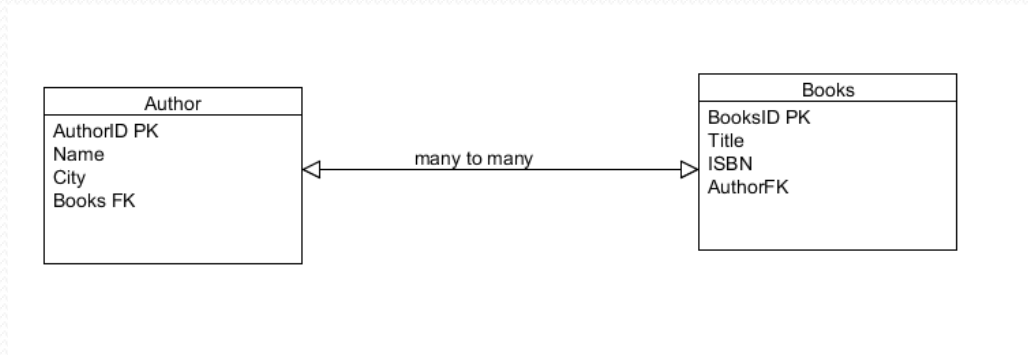
1:1 Relationship

- Not very common unless splitting large table into smaller tables
- Generally can use if certain columns in a table are rarely used



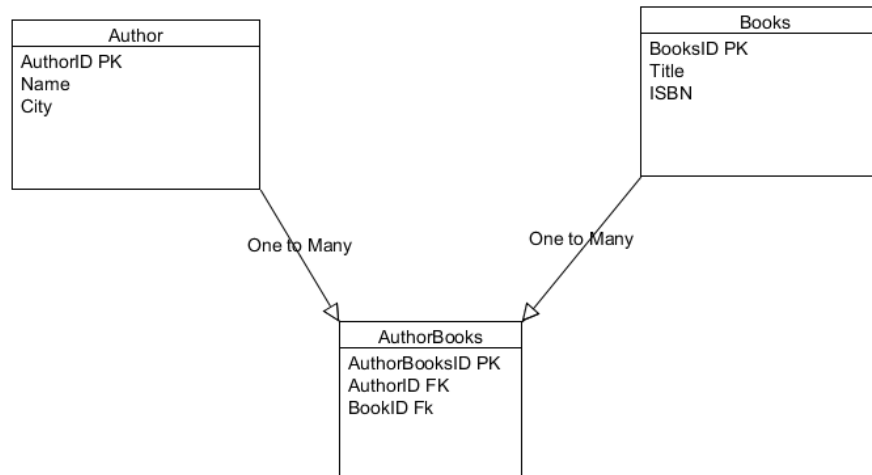
n:n Relationship

- Possible, however most DBA's will break a many to many relationship into two 1:n relationships
- The tables will use a junction table to enforce efficiency in the design



n:n Relationship(cont.)

A junction table will increase efficiency



Normalization

- Minimize data redundancy, reduce duplicate data
- Tables should be atomic(granular) in nature
- Only store information related to that table
- Each table should have their own unique identifier

1NF

- Table should be atomic in nature, meaning break up columns into granular values.
 - Example, name column can be broken up to FirstName and LastName
 - Address column can be broken up to Street, City, State, and Zip

| | | | | |
|--|----------|------|------|-----------------|
| | | | | |
| | Customer | | | |
| | ID | Name | City | CustomerContact |
| | | | | |
| | | | | |
| | | | | |
| | | | | |
| | | | | |
| | | | | |

How can we make this table conform to the 1NF?

2NF

- Any non-key field should be dependent on entire primary key
- Must satisfy 1NF to qualify for 2NF
- A table that is in 1st normal form and contains only a single key as the primary key is automatically in 2nd normal form.

Composite Key



| PurchaseDetail | | |
|----------------|---------|------------------|
| CustomerID | StoreID | PurchaseLocation |
| | | |
| | | |
| | | |
| | | |
| | | |
| | | |
| | | |
| | | |

PurchaseLocation is not dependent on entire PK

2NF

Breaking up the table will get rid of non-dependencies

| Purchase | | Store | |
|------------|---------|---------|------------------|
| CustomerID | StoreID | StoreID | PurchaseLocation |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |

3NF

- No non-key should be dependent on another non-key field
- Table should comply and satisfy rules of 1NF and 2NF

| OrderItems | | | | |
|--------------|-----------|-----------|----------|-------|
| OrderItemsID | ProductID | UnitPrice | Quantity | Total |
| | | | | |
| | | | | |
| | | | | |
| | | | | |
| | | | | |

Total does not make sense since it depends on UnitPrice and Quantity. You could have an illogical value added

Denormalization

- Adding redundancy and duplication
- Normalized data takes less space, but may require a join predicate to construct the desired result set, takes longer to read(query) data
- Denormalized data is replicated in several places. It then takes more space, but the result set of a query takes less times
- this improves **reading data** (because data is readily available), but **updating data** becomes more costly (because you need to update the replicated data).

Constraints

- Constraints let you define the way the Database Engine automatically enforces the integrity of a database. Constraints define rules regarding the values allowed in columns and are the standard mechanism for enforcing integrity
- The classes of constraints are:
 - UNIQUE
 - NOT NULL
 - CHECK
 - PK
 - FK – Referential Integrity

Microsoft SQL Server

- In this course will be working primarily with SQL Server 2012
- SQL Server Management Studio ships with SQL Server and is the IDE component of SQL Server
- SSMS vs Visual Studio, how are they similar?
- Let's take a tour of SMSS and look at the features that will allow us to build databases, tables, and columns. As well as work with reading and writing data

SQL Server Types

- SQL Server utilizes primitive types like .NET. A few may look familiar.
 - <http://msdn.microsoft.com/en-us/library/ms187752.aspx>
 - nvarchar and varchar

| | |
|----------------------|---------------------------|
| Exact numerics | Unicode character strings |
| Approximate numerics | Binary strings |
| Date and time | Other data types |
| Character strings | |

Building a sample Database

- Now we can try to build our own database and normalize it to the 3NF
- Think about real world objects, what data types your attributes will be, what keys you will need, and what kind of relationships you will set up
- Remember to Normalize your tables!

Restoring/Backing up a Database to SQL Server

- In this course we will be using Adventureworks and our own custom database
- Let's look at how to restore a database from a backup file to SQL so we can use it in SSMS
- Once we finish that, we also will learn how to backup an existing database

Structured Query Language

- Structured Query Language (SQL) is used to communicate with a database.
- According to ANSI (American National Standards Institute), it is the standard language for relational database management systems.
- SQL statements are used to perform tasks such as update data on a database, or retrieve data from a database.
- Common relational database management systems that use SQL are: Oracle, Sybase, Microsoft SQL Server, DB2, etc.

Working with SQL Statements

Data Definition Language (DDL)

- DDL statements are used to build and modify the structure of your tables and other objects in the database. When you execute a DDL statement, it takes effect immediately.
 - CREATE TABLE <table name> (
 <attribute name 1> <data type 1>,
 <attribute name 2> <data type 2>);
 - ALTER TABLE <table name>
 ADD CONSTRAINT <constraint name> PRIMARY KEY (<attribute list>);
 - DROP TABLE <table name>

SQL Statements

- Data Manipulation Language
 - Used for inserting, deleting and updating data in a database Performing read-only queries of data is also considered a component of DML.
 - Examples of DML are:
 - INSERT, UPDATE, DELETE
- Data Query Language
 - Used for querying data
 - SELECT

Practice with DDL

- In SSMS select new query and use the DDL statements to create a new DB and a table with no PK
- Use the ALTER statement to add a PK constraint
- Use the DROP statement to delete the table

SELECT Statement

- Note* SQL is not case sensitive
- Select – extracts data from a DB
 - Select * From Table
 - SELECT column_name(s)
FROM table_name
 - SELECT column_name(s)
FROM table_name
- Where – is used to filter records
 - SELECT column_name(s)
FROM table_name
WHERE column_name = 'value'
 - SELECT * FROM Person WHERE City='DC'
- SELECT * FROM Persons
WHERE FirstName='Joseph'
AND LastName='Yates'
- SELECT * FROM Persons
WHERE FirstName='John'
OR LastName='Yates'
- SELECT * FROM Persons WHERE
LastName='Smith'
AND (FirstName='Joe' OR FirstName='John')

Order By Statement

- Order By – used to sort the result-set by a specified column
 - `SELECT column_name(s)`
`FROM table_name`
`ORDER BY column_name(s) ASC|DESC`
 - `Select * From Persons`
`Order By LastName`

INSERT INTO Statement

- Insert Into – used to insert a new row into a table
 - INSERT INTO table_name
VALUES (value1, value2, value3,...)
 - INSERT INTO table_name (column1, column2, column3,...)
VALUES (value1, value2, value3,...)

UPDATE Statement

- Update – used to update records in a table
 - UPDATE table_name
SET column1=value, column2=value2,...
WHERE some_column=some_value
 - *be careful when using update, make sure to specify where clause or your entire table will be updated
 - UPDATE Persons
SET Address='55 Fairfax Blvd', City='Fairfax'
WHERE LastName='Mike' AND FirstName='Joseph'

LIKE Operator

- The LIKE operator is used to list all rows in a table whose column values match a specified pattern. It is useful when you want to search rows to match a specific pattern, or when you do not know the entire value. For this purpose we use a wildcard character '%'.
- ```
SELECT first_name, last_name
FROM customer
WHERE first_name LIKE 'S%';
```
- “%” signifies multiple characters and “\_” specifies single character.



# BETWEEN Operator

- The operator BETWEEN and AND, are used to compare data for a range of values.
- ```
SELECT first_name, last_name, age  
FROM student_details  
WHERE age BETWEEN 10 AND 15;
```

IN Operator

- The IN operator is used when you want to compare a column with more than one value. It is similar to an OR condition.
- ```
SELECT first_name, last_name, subject
FROM student_details
WHERE subject IN ('Math', 'Science');
```
- *NOTE:* The data used to compare is case sensitive

# DELETE Statement

- Delete – used to delete records in a table
  - DELETE FROM table\_name  
WHERE some\_column=some\_value
  - DELETE \* FROM table\_name

# SQL Functions

- Aggregate functions
  - COUNT(), MAX(), MIN(), AVG(), SUM()
- SQL Functions
  - getDate(), @@language, current\_User
- String Functions
  - UPPER(), Len()

# Stored Procedures

- Named batch of T-SQL Statements
- Functions – SELECT only
  - Use as part of other SQL statements
- SP's
  - Execute directly
  - INSERT, UPDATE, DELETE
  - Multiple batches of SQL
  - Input parameters
  - Output parameters

# Creating Stored Procedures

```
Use AdventureWorks
```

```
GO
```

```
CREATE PROCEDURE GetEmployees
```

```
@LastName nvarchar(50)
```

```
@FirstName nvarchar(50)
```

```
AS
```

```
SELECT FirstName, LastName,
```

```
FROM Employees
```

```
Where FirstName = @FirstName AND LastName = @LastName
```

# Execute a Stored Procedure

- EXEC GetEmployees @LastName = 'Doe', @FirstName = 'John'

OR ---

Execute GetEmployees @LastName = 'Doe', @FirstName = 'John'

- Try writing and executing a couple of stored procedures
- How would you modify a Stored Procedure???

# Advantages to Stored Procedures

- **Security:** Users can execute a stored procedure without needing to execute any of the statements directly. Therefore, a stored procedure can provide advanced database functionality for users who wouldn't normally have access to these tasks, but this functionality is made available in a tightly controlled way.
- **Faster execution:** Stored procedures are parsed and optimized as soon as they are created and the stored procedure is stored in memory. This means that it will execute a lot faster than sending many lines of SQL code from your application to the SQL Server. Doing that requires SQL Server to compile and optimize your SQL code every time it runs.
- **Reduced network traffic:** If you send many lines of SQL code over the network to your SQL Server, this will have an impact on network performance. This is especially true if you have hundreds of lines of SQL code and/or you have lots of activity in your application. Running the code on the SQL Server (as a stored procedure) eliminates the need to send this code over the network.
- **Reusable:** You can write a stored procedure once, then call it from multiple places in your application



# Transactions

- Think about a banking system
  - Move money from checking into savings
  - First you need to subtract from checking then add to savings
  - Both events must occur for transaction to succeed
- ACID properties
  - Atomic – must all happen or none occur
  - Consistent – can not violate any integrity rules, the database is left in a stable state at the end of the transaction
  - Isolated – data is locked, system can not access that data
  - Durable – robust data, survivability, data endures
- Implicit vs Explicit

# Transactions(cont.)

```
BEGIN TRAN
DECLARE @Discount float
SET @Discount = 0.05
UPDATE Products
SET UnitPrice = UnitPrice - (@Discount * UnitPrice)
Go
-- Check the results...
Select
ProductID, UnitPrice
FROM Products
ORDER BY UnitPrice Desc
If we don't like what we see, we can roll back the transaction and leave the database in its previous state
ROLLBACK
GO
SELECT
ProductID, UnitPrice -- See the original prices!
FROM Products
ORDER BY UnitPrice Desc
GO
Conversely to make the changes permanent we use the COMMIT operator:
COMMIT
Go
```

## Transactions(cont.)

- COMMIT: to save the changes.
- ROLLBACK: to rollback the changes.
- SAVEPOINT: creates points within groups of transactions in which to ROLLBACK

BEGIN TRAN

```
UPDATE authors
SET au_fname = 'John'
WHERE au_id = '172-32-1176'
```

```
UPDATE authors
SET au_fname = 'JohnY'
WHERE city = 'Lawrence'
```

```
IF @@ROWCOUNT = 5
 COMMIT TRAN
ELSE
 ROLLBACK TRAN
```

SQL supports If/Else conditional statements. @@ROWCOUNT returns an int



# Error Handling in Transactions

- The @@ERROR automatic variable is used to implement error handling code. It contains the error ID produced by the last SQL statement
- When a statement executes successfully, @@ERROR contains 0
  - To determine if a statement executes successfully, an IF statement is used to check the value of @@ERROR immediately after the target statement executes
  - @@ERROR must be checked immediately after the target statement, because its value is reset to 0 when the next statement executes successfully. If a trappable error occurs, @@ERROR will have a value greater than 0.
  - usually you'll want to test for changes in @@ERROR right after any INSERT, UPDATE, or DELETE statement.

# Error Handling in Transactions(cont.)

- CREATE PROCEDURE addTitle(@titleID VARCHAR(6), @authorID VARCHAR(11),  
 • @title VARCHAR(20), @titletype CHAR(12)) ← Transactions are usually nested in SP's
- AS
- BEGIN TRAN
- INSERT titles(titleID, title, type)
- VALUES (@titleID, @title, @titletype)
- IF (@@ERROR <> 0) GOTO ERR\_HANDLER
- INSERT titleauthor(authorID, titleID)
- VALUES (@authorID, @titleID)
- IF (@@ERROR <> 0) GOTO ERR\_HANDLER
- COMMIT TRAN
- RETURN 0
- ERR\_HANDLER:
- PRINT 'Unexpected error occurred!'
- ROLLBACK TRAN
- RETURN 1

# Error Handling in Transactions(cont.)

- Try...Catch blocks are supported
  - Similar to C# constructs
  - each TRY block is associated with only one CATCH block.
- Error Functions return information about the error
  - ERROR\_NUMBER() returns the error number.
  - ERROR\_MESSAGE() returns the complete text of the error message. The text includes the values supplied for any substitutable parameters such as lengths, object names, or times.
  - ERROR\_SEVERITY() returns the error severity.
  - ERROR\_STATE() returns the error state number.
  - ERROR\_LINE() returns the line number inside the routine that caused the error.
  - ERROR\_PROCEDURE() returns the name of the stored procedure or trigger where the error occurred.

# Error Handling in Transactions(cont.)

```
CREATE PROCEDURE GetErrorInfo
AS
 SELECT
 ERROR_NUMBER() AS ErrorNumber,
 ERROR_SEVERITY() AS ErrorSeverity,
 ERROR_STATE() as ErrorState,
 ERROR_PROCEDURE() as ErrorProcedure,
 ERROR_LINE() as ErrorLine,
 ERROR_MESSAGE() as ErrorMessage;
GO

BEGIN TRY
 -- Generate divide-by-zero error.
 SELECT 1/o;
END TRY
BEGIN CATCH
 -- Execute the error retrieval routine.
 EXECUTE GetErrorInfo;
END CATCH;
GO
```

# Triggers

- Specialized stored procedure that executes on a DML or DDL statement
- DML trigger – enforces business rules
  - Insert trigger
  - Update trigger
  - Delete trigger
- FOR/AFTER trigger
  - After underlying data is modified
  - Can not be used in views
- INSTEAD OF trigger
  - Runs in place of DML statement
    - Ex. “Insert”



# Joins

- With joins you can retrieve data from two or more tables based on logical relationships between the tables
- a join condition defines the way two tables are related in a query by specifying the column from each table to be used for the join. A typical join condition specifies a foreign key from one table and its associated key in the other table usually a PK.
- joins are typically performed in the FROM clause of a table or view for the SELECT, INSERT...SELECT, SELECT...INTO, UPDATE and DELETE statements

# Inner Join

- Inner join returns only those records/rows that match/exists in both the tables.

--Companies and contacts with where we ship their items.

```
select CompanyName, ContactName, ContactTitle, ShipAddress, ShipCity
from Customers c inner join Orders o
on c.CustomerID = o.CustomerID
```

-- What categories products fall into and their description

```
select ProductName, UnitPrice, CategoryName, Description
from Products p inner join Categories c
on p.CategoryID = c.CategoryID
```

## Right Outer Join

- The RIGHT JOIN keyword returns all the rows from the right table, even if there are no matches in the left table. If there are no columns matching in the left table, it returns NULL values.

```
--Employees processing orders and when they were shipped.
select e.FirstName, e.LastName, o.OrderDate, o.ShippedDate
from Employees e right join Orders o
on e.EmployeeID = o.EmployeeID
```

# Left Outer Join

- The LEFT JOIN keyword returns all rows from the left table, even if there are no matches in the right table. If there are no columns matching in the right table, it returns NULL values.

```
select e.FirstName, e.LastName, o.ShipName
from Employees e left join Orders o
on e.EmployeeID = o.EmployeeID
```

```
Select c.CompanyName, c.ContactName, o.ShippedDate, o.Freight
from Customers c left join Orders o
on c.CustomerID = o.CustomerID
where o.Freight > 5
```

# Full Outer Join

- Based on the two tables specified in the join clause, all data is returned from both tables regardless of matching data

-- Products we sell and where/who we get them from.

```
select ProductName, UnitPrice, CompanyName, ContactName
from Products p full outer join Suppliers s
on p.SupplierID = s.SupplierID
where UnitPrice > 10
```

# Cross Join

- Cross join is a Cartesian join means Cartesian product of both the tables. This join does not need any condition to join two tables. This join returns records/rows that are multiplication of record number from both the tables means each row on left table will related to each row of right table

# Self Join

- Self join is used to retrieve the records having some relation or similarity with other records in the same table
- Assume we have an employee table, one of the columns in the same table contains the ID of a manager, who is also an employee for the same company. This way, all the employees and their managers are present in the same table. If we want to find the manager of a particular employee, we need to use a self join.
- Note - Self Joins are not actually their own type of joins they are Outer, Inner, or Cross joins.

# Self Joins

```
SELECT e1.Name EmployeeName,
 e2.name AS ManagerName
FROM Employee e1
INNER JOIN Employee e2
ON e1.ManagerID = e2.EmployeeID
```

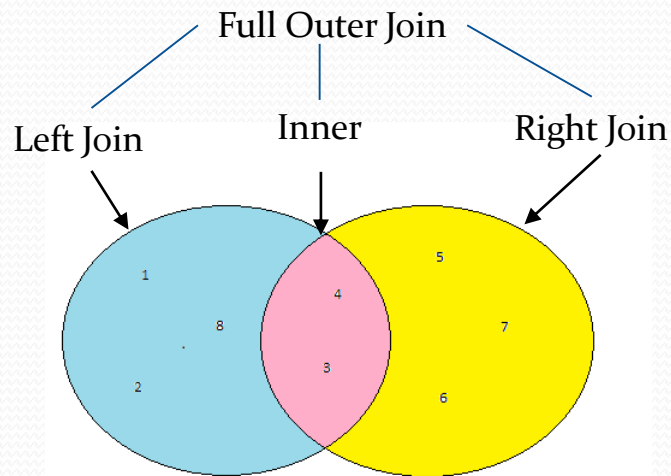
```
SELECT e1.Name EmployeeName,
 ISNULL(e2.name, 'Top Manager')
 AS ManagerName
FROM Employee e1
LEFT JOIN Employee e2
ON e1.ManagerID =
 e2.EmployeeID
```

Checks 1<sup>st</sup>  
parameter for  
Null, true returns  
2nd

Outer and Inner are  
performed as “Self Joins”



# Joins still confusing?



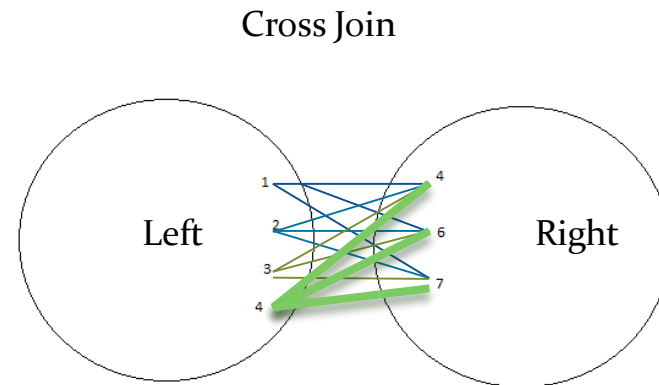
Inner Join Result : (4,3)

Left Join Result : (1,2,8,4,3)

Right Join Result : (5,6,7,4,3)

Full Outer Join Result : (1,2,8,4,3,5,6,7)

Cross Join Result : ((1,4), (1,6), (1,7), (2,4), (2,6), (2,7), (3,4), (3,6), (3,7), (4,4), (4,6), (4,7))



# UNION vs UNION ALL

- **UNION**
  - The UNION command is used to select related information from two tables, much like the JOIN command. However, when using the UNION command all selected columns need to be of the same data type. With UNION, only distinct values are selected.
- **UNION ALL**
  - The UNION ALL command is equal to the UNION command, except that UNION ALL selects all values.
- The difference between Union and Union all is that Union all will not eliminate duplicate rows, instead it just pulls all rows from all tables fitting your query and combines them into a table.
- A UNION statement effectively does a SELECT DISTINCT on the results set. If you know that all the records returned are unique from your union, use UNION ALL instead, it gives faster results.

# Views

- views can be thought of as either a virtual table or a stored query. The data accessible through a view is not stored in the database as a distinct object. What is stored in the database is a SELECT statement. The result set of the SELECT statement forms the virtual table returned by the view
- Virtual tables are used when security is a concern and can span multiple tables using joins
- Restrict a user to specific rows in a table.
  - Ex. allow an employee to see only the rows recording his or her work in a labor-tracking table.
- Restrict a user to specific columns.
  - Ex. allow employees who do not work in payroll to see the name, office, work phone, and department columns in an employee table, but do not allow them to see any columns with salary information or personal information.
- Join columns from multiple tables so that they look like a single table.
- Aggregate information instead of supplying details.
  - Ex. present the sum of a column, or the maximum or minimum value from a column

## Views(cont.)

- If we wanted to create a view that would encapsulate the details of an employee who had all their personal information, like salary and SSN, we could use a view. Then if an end user needed to access the info of an employee, like name or work number, they could so through a view

```
CREATE VIEW EmployeeInfo
```

```
AS
```

```
SELECT FirstName, LastName, WorkNumber
```

```
FROM Employees
```

- Then you could use a SELECT statement to query the View

```
Select * from EmployeeInfo
```

# Indexes

- The most important route to high performance in a SQL Server database is the index. Indexes speed up the querying process by providing swift access to rows in the data tables, similarly to the way a book's index helps you find information quickly within that book.
- Indexes are created on columns in tables or views. The index provides a fast way to look up data based on the values within those columns. For example, if you create an index on the primary key and then search for a row of data based on one of the primary key values, SQL Server first finds that value in the index, and then uses the index to quickly locate the entire row of data. Without the index, a table scan would have to be performed in order to locate the row, which can have a significant effect on performance.
- Data is arranged by SQL Server in the form of *extents* and *pages*. Each extent is of size 64 KB, having 8 pages of 8KB sizes. An extent may have data from multiple or same table, but each page holds data from a single table only
- What is a B-tree?

## Indexes(cont.)

- A table is nothing but a collection of record sets; by default, rows are stored in the form of heaps unless a clustered index has been defined on the table, in which case, record sets are sorted and stored on the clustered index. The heap's structure is a simple arrangement where the inserted record is stored in the next available space on the table page
- Heaps are good at storing data, however heaps are inefficient when a large amount of reads are being performed
- Indexes are arranged in the form of a B-Tree where the leaf node holds the data or a pointer to the data. Since the stored data is in a sorted order, indexes precisely know which record is sitting where. Hence an index optimizes and enhances the data retrieval immensely
- Every time there is a DML (insert/update/delete) fired on an indexed table, SQL Server updates the index to be able to identify the record. Hence if there are more indexes, it's liable that the DMLs will take a longer time to execute

# Indexes(cont.)

- Clustered Index
  - clustered index will be created on a table by default the moment a primary key is created on the table
  - A table can only have one CI
  - key-value pair in the clustered index has the index key and the actual data value
  - If a table has no clustered index, its data rows are stored in an unordered structure called a heap.
- Non-Clustered Index
  - Nonclustered indexes have a structure separate from the data rows. A nonclustered index contains the nonclustered index key values and each key value entry has a pointer to the data row that contains the key value.
  - The pointer from an index row in a nonclustered index to a data row is called a row locator. The structure of the row locator depends on whether the data pages are stored in a heap or a clustered table. For a heap, a row locator is a pointer to the row. For a clustered table, the row locator is the clustered index key

# SQL Server Administration

- SQL Profiler is a tool provided by SQL Server for DBA's/Developers to analyze queries
  - a rich interface to create and manage traces and analyze and replay trace results
  - events are saved in a trace file that can later be analyzed or used to replay a specific series of steps when trying to diagnose a problem.
- Backup/Restore
  - Use a .bak file to restore a database to SQL Server, also can create a .bak file from a database
  - Use a .mdf file to attach an existing database to SQL Server



# Agenda

- Introduction
- Types of Languages
- DDL and DML
- Constraints
- Select Statements
- Clauses
- Joins

# Introduction

- Structured Query Language (SQL) is used to communicate with a database.
- According to ANSI (American National Standards Institute), it is the standard language for relational database management systems.
- SQL statements are used to perform tasks such as update data on a database, or retrieve data from a database.
- Common relational database management systems that use SQL are: Oracle, Sybase, Microsoft SQL Server, DB2, etc.

# Types of Languages

- DDL – Data Definition Language
  - create, alter and drop
- DML – Data Manipulation Language
  - insert, update and delete
- DQL – Data Query Language
  - select
- TCL – Transaction Control Language
  - commit, rollback and savepoint

# DDL

- Data Definition Language (DDL) statements are used to define the database structure or schema.
- CREATE
  - to create objects in the database like Table
- ALTER
  - alters the structure of the database table
- DROP
  - delete objects like table from the database

# Tables

- A relational database system contains one or more objects called tables.
- The data or information for the database are stored in these tables.
- Tables are uniquely identified by their names and are comprised of columns and rows.
- Columns contain the column name, data type, and any other attributes for the column.
- Rows contain the records or data for the columns.

# Constraints

- SQL constraints are rules that you define which data values are valid while doing INSERT, UPDATE, and DELETE operations.
- When constraints are in place, SQL engine rejects all the transactions that break the rules therefore constraints help you enforce the integrity of data automatically.
- SQL provides five types of constraints
  - PRIMARY KEY
  - NOT NULL
  - UNIQUE
  - FOREIGN KEY
  - CHECK

# Constraints

- Primary key
- This constraint defines a column or combination of columns which uniquely identifies each row in the table.
- Foreign key or Referential Integrity
- This constraint identifies any column referencing the PRIMARY KEY in another table. It establishes a relationship between two columns in the same table or between different tables.
- Not Null
- This constraint ensures all rows in the table contain a definite value for the column which is specified as not null. Which means a null value is not allowed.

# Constraints

- Unique key
- This constraint ensures that a column or a group of columns in each row have a distinct value. A column(s) can have a null value but the values cannot be duplicated.
- Check
- This constraint defines a business rule on a column. All the rows must satisfy this rule. The constraint can be applied for a single column or a group of columns.



# DML

- Data Manipulation Language (DML) statements are used for managing data within schema objects.
- INSERT
  - insert data into a table
- UPDATE
  - updates existing data within a table
- DELETE
  - deletes specific or all records from a table

# Insert statement

- The **insert** statement is used to insert or add a row of data into the table.
- To insert records into a table, enter the key words **insert into** followed by the table name, followed by the keyword **values**, followed by the list of values enclosed in parenthesis.
- Strings should be enclosed in single quotes, and numbers should not.
- ```
insert into "tablename"  
    (first_column,...last_column)  
    values (first_value,...last_value);
```

Update statement

- The **update** statement is used to update or change records that match a specified criteria.
- This is accomplished by carefully constructing a where clause.
- ```
update "tablename"
 set "columnname" = "newvalue"
 [,"nextcolumn" = "newvalue2"...]
 where "columnname"
 OPERATOR "value"
 [and|or "column"
 OPERATOR "value"];
```

# Delete statement

- The **delete** statement is used to delete records or rows from the table.
- delete from "tablename"  
    where "columnname"  
    OPERATOR "value"  
    [and|or "column"  
    OPERATOR "value"];
- To delete an entire record/row from a table, enter "delete from" followed by the table name, followed by the where clause which contains the conditions to delete. If you leave off the where clause, all records will be deleted.

# Drop table

- The **drop table** command is used to delete a table and all rows in the table.
- To delete an entire table including all of its rows, issue the **drop table** command followed by the tablename.
- **drop table** is different from deleting all of the records in the table. Deleting all of the records in the table leaves the table including column and constraint information.
- Dropping the table removes the table definition as well as all of its rows.
- `drop table "tablename"`

# TCL

- Transaction control statements manage changes made by DML statements.
- A transaction is a set of SQL statements which is treated as a Single Unit. i.e. all the statements should execute successfully or none.
- COMMIT
  - Make changes done in transaction permanent.
- ROLLBACK
  - Revert the state of database to the last commit point.
- SAVEPOINT
  - Use to specify a point in transaction to which later you can rollback.

# Select statement

- The **select** statement is used to query the database and retrieve selected data that match the criteria that you specify.
- You can select as many column names that you'd like, or you can use a "\*" to select all columns.
- The table name that follows the keyword **from** specifies the table that will be queried to retrieve the desired results.
- The **where** clause (optional) specifies which data values or rows will be returned or displayed, based on the criteria described after the keyword **where**.
- ```
select "column1" [,"column2",etc]  
    from "tablename"  
    [where "condition"];
```

Select statement

- Conditional selection used in **where** clause

= Equal

> Greater than

< Less than

>= Greater than or equal

<= Less than or equal

<> Not equal to

Aggregate functions

| | |
|----------|---|
| MIN | returns the smallest value in a given column |
| MAX | returns the largest value in a given column |
| SUM | returns the sum of the numeric values in a given column |
| AVG | returns the average value of a given column |
| COUNT | returns the total number of values in a given column |
| COUNT(*) | returns the number of rows in a table |

Where Clause

- The WHERE Clause is used when you want to retrieve specific information from a table excluding other irrelevant data.
- The condition provided in the WHERE clause filters the rows retrieved from the table and gives only those rows which is expected.
- ```
select "column1" [,"column2",etc]
 from "tablename"
 [where "condition"];
```
- WHERE clause can be used along with SELECT, DELETE, UPDATE statements.

# Group by clause

- The GROUP BY clause will gather all of the rows together that contain data in the specified column(s) and will allow aggregate functions to be performed on the one or more columns.
- ```
SELECT column1,  
SUM(column2)  
FROM "list-of-tables"  
GROUP BY "column-list";
```

Having clause

- The HAVING clause allows you to specify conditions on the rows for each group - in other words, which rows should be selected will be based on the conditions you specify.
- The HAVING clause should follow the GROUP BY clause if you are going to use it.
- ```
SELECT column1,
SUM(column2)
FROM "list-of-tables"
GROUP BY "column-list"
HAVING "condition";
```

# Order by clause

- ORDER BY is an optional clause which will allow you to display the results of your query in a sorted order (either ascending order or descending order) based on the columns that you specify to order by.
- If you would like to order based on multiple columns, you must separate the columns with commas.
- ```
SELECT column1,  
SUM(column2)  
FROM "list-of-tables"  
ORDER BY "column-list" [ASC | DESC];
```

Like Operator

- The LIKE operator is used to list all rows in a table whose column values match a specified pattern. It is useful when you want to search rows to match a specific pattern, or when you do not know the entire value. For this purpose we use a wildcard character '%'.
- ```
SELECT first_name, last_name
FROM customer
WHERE first_name LIKE 'S%';
```
- “%” signifies multiple characters and “\_” specifies single character.

# Between Operator

- The operator BETWEEN and AND, are used to compare data for a range of values.
- ```
SELECT first_name, last_name, age  
FROM student_details  
WHERE age BETWEEN 10 AND 15;
```

In Operator

- The IN operator is used when you want to compare a column with more than one value. It is similar to an OR condition.
- ```
SELECT first_name, last_name, subject
FROM student_details
WHERE subject IN ('Maths', 'Science');
```
- *NOTE:* The data used to compare is case sensitive.



# Sequence

- A sequence is a user-defined schema bound object that generates a sequence of numeric values according to the specification with which the sequence was created.
- `CREATE SEQUENCE seqname [ INCREMENT increment ]  
[ MINVALUE minvalue ] [ MAXVALUE maxvalue ]  
[ START start ]`

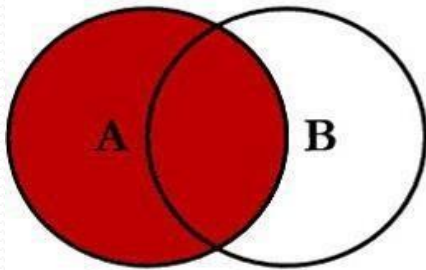
# Joins

- It is one of the most beneficial features of SQL & relational database systems - the "**Join**".
- The "Join" makes relational database systems "relational".
- Joins allow you to link data from two or more tables together into a single query result--from one single SELECT statement.
- A "Join" can be recognized in a SQL SELECT statement if it has more than one table after the FROM keyword.
- ```
SELECT "list-of-columns"  
FROM table1,table2  
WHERE "search-condition(s)"
```

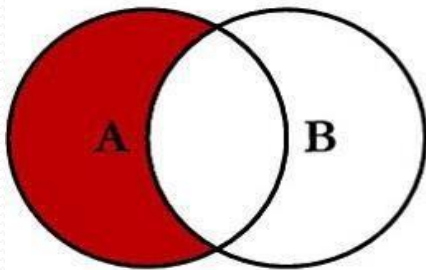
Joins

- Equi joins
 - It is a simple sql join condition which uses the equal sign as the comparison operator. Two types of equi joins are SQL Outer join and SQL Inner join.
- Inner Join
 - All the rows returned by the sql query satisfy the sql join condition specified.
- Outer Join
 - This sql join returns all rows from both tables which satisfy the join condition along with rows which do not satisfy the join condition from one of the tables.
- Non equi joins
 - It is a sql join condition which makes use of some comparison operator other than the equal sign like $>$, $<$, $>=$, $<=$

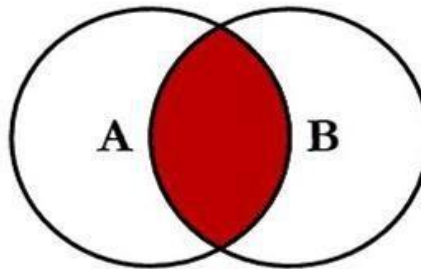
SQL JOINS



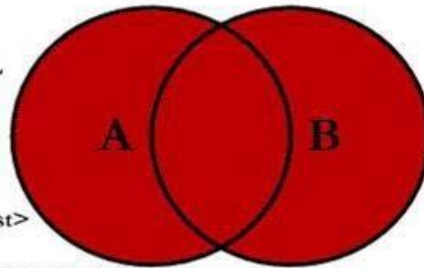
```
SELECT <select_list>
FROM TableA A
LEFT JOIN TableB B
ON A.Key = B.Key
```



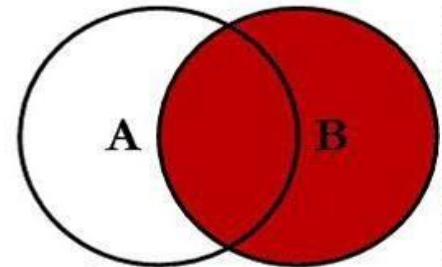
```
SELECT <select_list>
FROM TableA A
LEFT JOIN TableB B
ON A.Key = B.Key
WHERE B.Key IS NULL
```



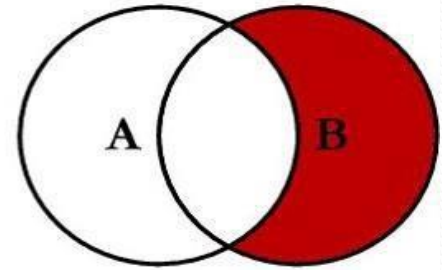
```
SELECT <select_list>
FROM TableA A
INNER JOIN TableB B
ON A.Key = B.Key
```



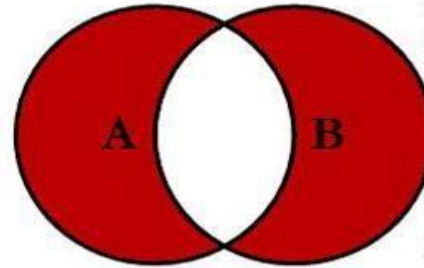
```
SELECT <select_list>
FROM TableA A
FULL OUTER JOIN TableB B
ON A.Key = B.Key
```



```
SELECT <select_list>
FROM TableA A
RIGHT JOIN TableB B
ON A.Key = B.Key
```



```
SELECT <select_list>
FROM TableA A
RIGHT JOIN TableB B
ON A.Key = B.Key
WHERE A.Key IS NULL
```



```
SELECT <select_list>
FROM TableA A
FULL OUTER JOIN TableB B
ON A.Key = B.Key
WHERE A.Key IS NULL
OR B.Key IS NULL
```

PL/SQL

Introduction

- PL/SQL stands for Procedural Language extension of SQL.
- PL/SQL is a combination of SQL along with the procedural features of programming languages.
- PL/SQL is used for defining Procedures & Functions
- PL/SQL is also used for Cursors and Triggers

Stored Procedures

- A stored procedure is a named PL/SQL block which performs one or more specific task. This is similar to a procedure in other programming languages.
- We can pass parameters to procedures in two ways.
 - IN-parameters: Parameters taken as arguments
 - OUT-parameters: Parameters giving values back.
- ```
CREATE [OR REPLACE] PROCEDURE proc_name [list of parameters] IS
 Declaration section
BEGIN
 Execution section
END;
/
```

# Creating a Stored Procedure in Oracle 10g XE

ORACLE Database Express Edition

User: TRAINER

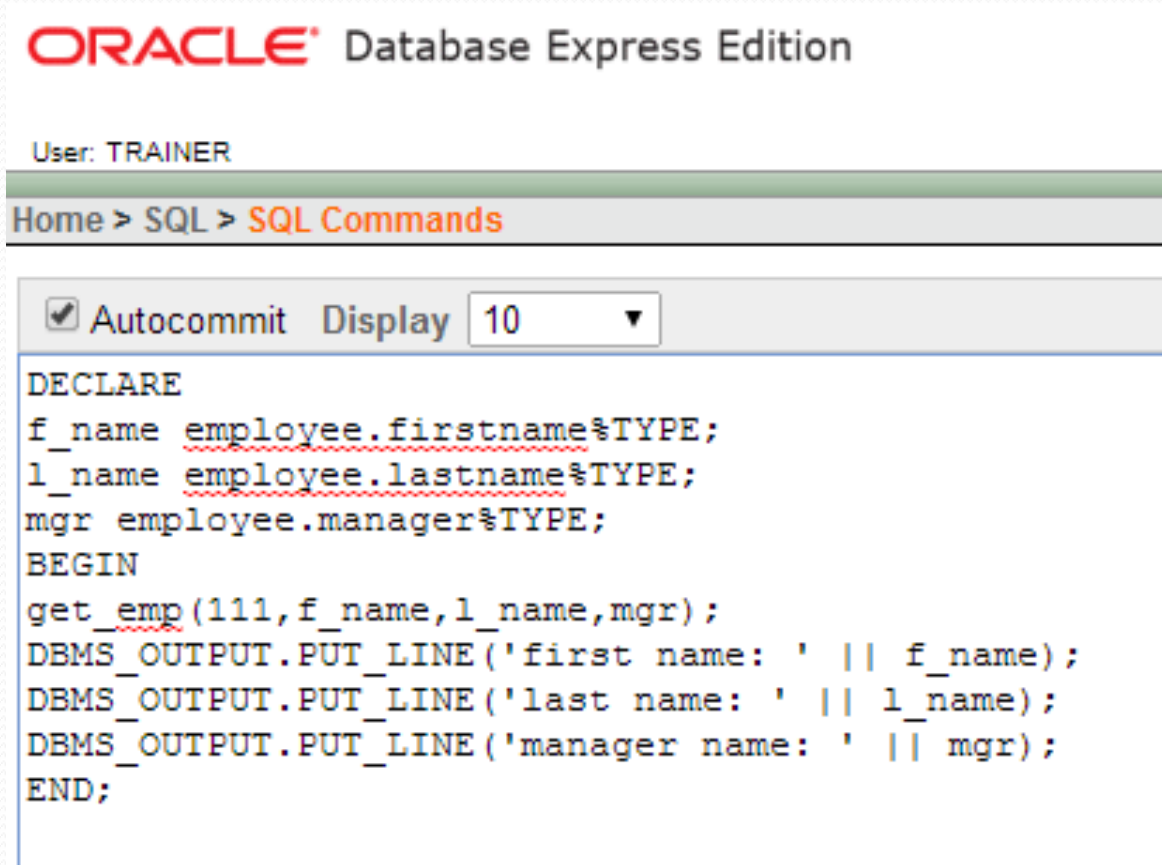
Home > SQL > SQL Commands

☒ Autocommit Display 10 ▼

```
create or replace procedure insert_emp(e_id IN NUMBER, e_fname IN VARCHAR, e_lname IN VARCHAR, e_manager IN VARCHAR)
AS BEGIN
INSERT into employee(empid, firstname, lastname, manager) values(e_id, e_fname, e_lname, e_manager);
commit;
END insert_emp;
```



# Calling a Stored Procedure in Oracle 10g XE



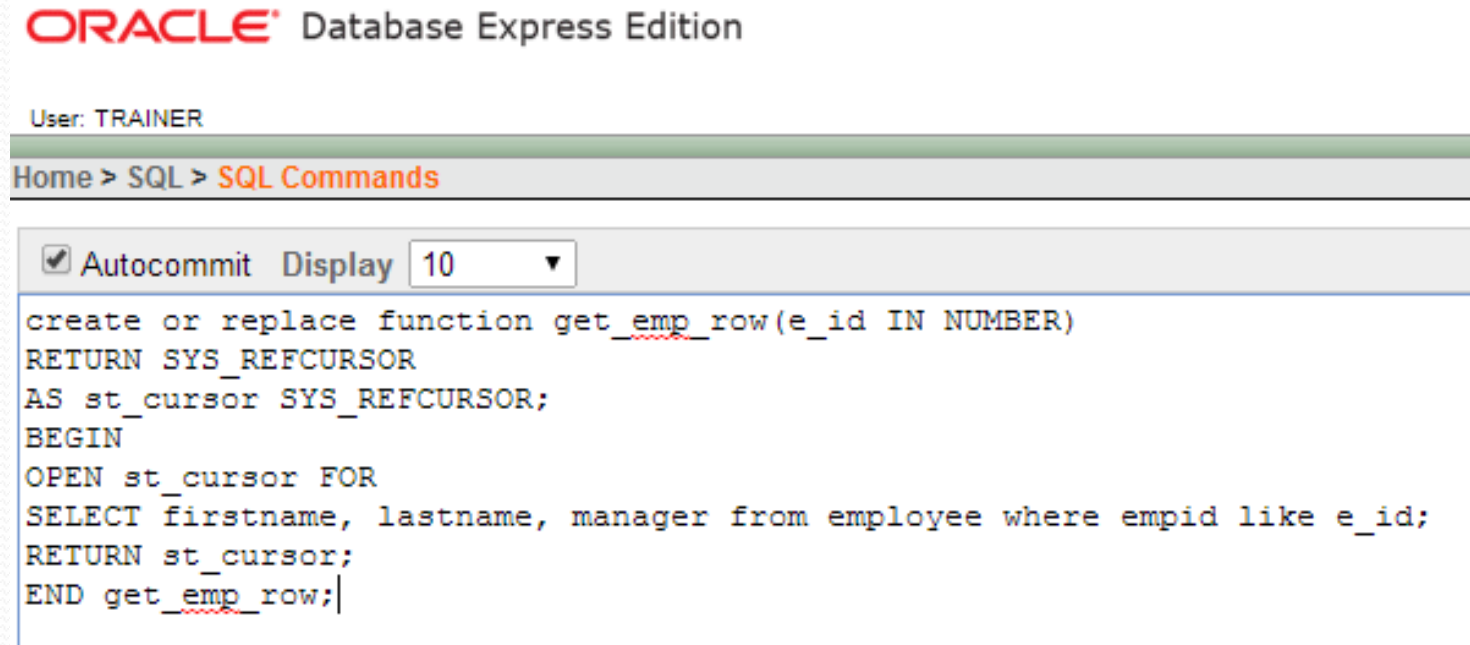
The screenshot shows the Oracle Database Express Edition web interface. At the top, the text "ORACLE Database Express Edition" is displayed. Below this, the user is identified as "User: TRAINER". A breadcrumb trail shows "Home > SQL > SQL Commands". A control bar includes a checked "Autocommit" checkbox and a "Display" dropdown menu set to "10". The main area contains a SQL script for calling a stored procedure.

```
DECLARE
f_name employee.firstname%TYPE;
l_name employee.lastname%TYPE;
mgr employee.manager%TYPE;
BEGIN
get_emp(111,f_name,l_name,mgr);
DBMS_OUTPUT.PUT_LINE('first name: ' || f_name);
DBMS_OUTPUT.PUT_LINE('last name: ' || l_name);
DBMS_OUTPUT.PUT_LINE('manager name: ' || mgr);
END;
```

# Functions

- A function is a named PL/SQL Block which is similar to a procedure. The major difference between a procedure and a function is, a function must always return a value, but a procedure may or may not return a value.
- ```
CREATE [OR REPLACE] FUNCTION function_name [parameters]
RETURN return_datatype IS
    Declaration_section
BEGIN
    Execution_section
    Return return_variable;
END;
/
```

Creating a Function in Oracle 10g XE



The screenshot shows the Oracle Database Express Edition web interface. At the top, it says "ORACLE Database Express Edition". Below that, it indicates the user is "TRAINER". A breadcrumb trail shows "Home > SQL > SQL Commands". There is a control bar with a checked "Autocommit" checkbox and a "Display" dropdown set to "10". The main area contains the following SQL code:

```
create or replace function get_emp_row(e_id IN NUMBER)
RETURN SYS_REFCURSOR
AS st_cursor SYS_REFCURSOR;
BEGIN
OPEN st_cursor FOR
SELECT firstname, lastname, manager from employee where empid like e_id;
RETURN st_cursor;
END get_emp_row;
```

Other PL/SQL Blocks

- Cursor
 - A cursor is a temporary work area created in the system memory when a SQL statement is executed. A cursor contains information on a select statement and the rows of data accessed by it.
- Trigger
 - A trigger is a PL/SQL block structure which is fired when a DML statements like Insert, Delete, Update is executed on a database table. A trigger is triggered automatically when an associated DML statement executes.
 - Row level trigger
 - An event is triggered for each row updated, inserted or deleted.
 - Statement level trigger
 - An event is triggered for each sql statement executed.

End