Agenda

- SQL Overview
- SQL – DDL
  - CREATE, DROP, ALTER
- SQL – DML
  - SELECT, INSERT, UPDATE, DELETE
- Subqueries, Joins, Unions, Intersections and Difference
SQL Overview

- What is SQL?
  - Structured Query Language
  - SQL is descriptive (what to get), not a procedural language (how to get)
  - It is a SET based language
  - It is the real world implementation of Relational Algebra

- SQL-99, which includes Object-Oriented Data Management, and some procedural commands is being implemented in vendor’s own flavors
SQL Overview (Continued)

- SQL
  - DDL: Data Definition Language
    - Create, Alter, Drop
  - DML: Data Manipulation Language
    - Select, Insert, Update, Delete
Data Definition Language (DDL)

- SQL DDL is used for creating and destroying DB objects: – Domains, Tables, Views, Indexes…
  - Main SQL DDL statements are:
    - CREATE/ALTER TABLE
    - CREATE VIEW
    - CREATE INDEX
    - CREATE/ALTER DOMAIN
    - DROP TABLE
    - DROP VIEW
    - DROP INDEX
    - DROP DOMAIN
CREATE TABLE Student (sid INTEGER,
    name CHAR(30),
    login CHAR(20),
    age INTEGER,
    gpa REAL,
CONSTRAINT student_key PRIMARY KEY (sid));
Oracle Data Types

- **CHAR(n):** fixed n char (default = 1; max n= 255)
- **VARCHAR2(n):** varying chars (max=2000)
- **DATE:** ‘DD-MON-YY’ or ‘DD-MON-YYYY’
- **NUMBER (n):** integer or real values (default = 40)
- **NUMBER (n,d):** max n spaces with d digits after decimal point
- **INTEGER:** same as NUMBER, but no decimal digits
- **INTEGER(n):** Integer with n spaces
- **LONG:** up to 2GB characters
- **BLOB:** up to 4GB of binary large objects
- **CLOB:** up to 4GB of character large objects
SQL – DML: SELECT

- **SELECT**
  - THE most important statement in the language, with maximum options/variations to it.
  - Implements 3 relation algebra operations
    - Selection, Projection, Join
SQL – DML: SELECT (Continued)

- Parts of SELECT statement
  - SELECT clause: For Selection
  - FROM clause: Which ‘Relation/Relations’ to use
  - WHERE clause: For Join conditions, and Filtering
  - ORDER BY: For sorting the output
  - GROUP BY: For creating ‘data buckets’
  - HAVING: For ‘filtering’ the groups
WHERE (used to specify search conditions)

- Comparison
  - =, >, <, <>, NOT, AND, OR

- Range Search
  - BETWEEN, NOT BETWEEN

- Set Membership search
  - IN, NOT IN

- Pattern match search condition
  - LIKE, NOT LIKE

- NULL search conditions
  - IS NULL, IS NOT NULL
SQL – SELECT

- COLUMN FUNCTIONS
  - COUNT
  - SUM
  - AVG
  - MIN
  - MAX

- Example:
  - SELECT AVG(e.age) FROM Employee e;
SQL – DML: INSERT

- INSERT – 2 formats
  - INSERT INTO
tablename[(column_list)]......VALUES
  (value_list)......
  - INSERT INTO
tablename[(column_list)]......AS SELECT......

INSERT INTO Student(sid, name, login, age, gpa)
VALUES (53688, ‘Smith’, ‘smith@ee’, 18, 3.2);
SQL – DML: UPDATE

- UPDATE
  - ALL rows: don’t use Where clause
  - Specific row(s) : Use Where clause

UPDATE Student S
SET S.age = S.age + 1
WHERE S.sid = 53688;
DELETE – 2 forms: Operates at Row level

- ALL rows: don’t use Where clause
- Specific row(s): Use Where clause

DELETE *
FROM Student S
WHERE S.sid = 53688;
SQL – DML: Combining Relations

- Combining Relations
  - SUBQUERIES
  - JOINS
    - Simple
    - Multiple relations/conditions
    - Outer Joins
      - Left Outer Join
      - Right Outer Join
      - Full Outer Join
  - UNION
  - INTERSECT
  - EXCEPT
Subqueries

- SQL statements with embedded SELECT.
- Can be used in
  - INSERT, UPDATE and DELETE
  - In WHERE and HAVING clauses of the outer SELECT, where it is called a *subquery* or *nested query*.
  - IN SELECT, FROM and WHERE clauses as well

```sql
SELECT S.name
FROM Student S
WHERE S.age > (SELECT MAX(S2.age) FROM Student S2 WHERE S2.gpa = 3.2);
```
Correlated Subqueries

- Columns in queries have scope, as do variables in a program
- Columns in the top-most query have “global scope”
- If one of these columns is used in a subquery or nested query, it is considered a correlated subquery.

BUT DON'T DO IT!
Correlated Subqueries - example

```
SELECT e.empid, e.salary
FROM emp e
WHERE e.dept =
  (SELECT dept
   FROM emp
   WHERE empid =
     (SELECT empid
      FROM invoice
      WHERE invoice_number = e.empid));
```
Subqueries – Aggregate Functions

List all staff whose salary is greater than the average salary, and show by how much.

```
SELECT staffNo, fName, lName, position, salary - (SELECT AVG(salary) FROM Staff) As SalDiff
FROM Staff
WHERE salary > (SELECT AVG(salary) FROM Staff);
```

- Cannot write ‘WHERE salary > AVG(salary)’
Subqueries – Aggregate Functions

- Instead, use subquery to find average salary (17000), and then use outer SELECT to find those staff with salary greater than this:

```
SELECT staffNo, fName, lName, position, salary - 17000 As salDiff
FROM Staff
WHERE salary > 17000;
```
Subquery Rules

- ORDER BY clause may not be used in a subquery (although it may be used in outermost SELECT).

- Subquery SELECT list must consist of a single column name or expression, except for subqueries that use EXISTS.

- By default, column names refer to table name in FROM clause of subquery. Can refer to a table in FROM using an alias.

- When subquery is an operand in a comparison, subquery must appear on right-hand side.

- A subquery may not be used as an operand in an expression.
Outer Joins

- If one row of an (inner) joined table is unmatched, then the row is omitted from result table.
- Outer join operations retain rows that do not satisfy the join condition.
- Consider following tables:

<table>
<thead>
<tr>
<th>Branch1</th>
<th>PropertyForRent1</th>
</tr>
</thead>
<tbody>
<tr>
<td>branchNo</td>
<td>propertyNo</td>
</tr>
<tr>
<td>bCity</td>
<td>pCity</td>
</tr>
<tr>
<td>B003</td>
<td>PA14</td>
</tr>
<tr>
<td>Glasgow</td>
<td>Aberdeen</td>
</tr>
<tr>
<td>B004</td>
<td>PL94</td>
</tr>
<tr>
<td>Bristol</td>
<td>London</td>
</tr>
<tr>
<td>B002</td>
<td>PG4</td>
</tr>
<tr>
<td>London</td>
<td>Glasgow</td>
</tr>
</tbody>
</table>
Outer Join

The (inner) join of these two tables:

```sql
SELECT b.*, p.*
FROM Branch1 b, PropertyForRent1 p
WHERE b.bCity = p.pCity;
```

- Result table has two rows where cities are same.
- There are no rows corresponding to branches in Bristol and Aberdeen.
- To include unmatched rows in result table, use an Outer join.

**Table 5.27(b)** Result table for inner join of Branch1 and PropertyForRent1 tables.

<table>
<thead>
<tr>
<th>branchNo</th>
<th>bCity</th>
<th>propertyNo</th>
<th>pCity</th>
</tr>
</thead>
<tbody>
<tr>
<td>B003</td>
<td>Glasgow</td>
<td>PG4</td>
<td>Glasgow</td>
</tr>
<tr>
<td>B002</td>
<td>London</td>
<td>PL94</td>
<td>London</td>
</tr>
</tbody>
</table>
Left Outer Join

- List branches and properties that are in same city along with any unmatched branches.
  
  ```sql
  SELECT b.*, p.*
  FROM Branch1 b LEFT OUTER JOIN PropertyForRent1 p ON b.bCity = p.pCity;
  ```

- Includes those rows of first (left) table unmatched with rows from second (right) table.
- Columns from second table are filled with NULLs.

<table>
<thead>
<tr>
<th>branchNo</th>
<th>bCity</th>
<th>propertyNo</th>
<th>pCity</th>
</tr>
</thead>
<tbody>
<tr>
<td>B003</td>
<td>Glasgow</td>
<td>PG4</td>
<td>Glasgow</td>
</tr>
<tr>
<td>B004</td>
<td>Bristol</td>
<td>NULL</td>
<td>NULL</td>
</tr>
<tr>
<td>B002</td>
<td>London</td>
<td>PL94</td>
<td>London</td>
</tr>
</tbody>
</table>
Right Outer Join

- List branches and properties in same city and any unmatched properties.
  
  ```sql
  SELECT b.*, p.*
  FROM Branch1 b RIGHT OUTER JOIN PropertyForRent1 p ON b.bCity = p.pCity;
  ```

- Right Outer join includes those rows of second (right) table that are unmatched with rows from first (left) table.

- Columns from first table are filled with NULLs.

<table>
<thead>
<tr>
<th>branchNo</th>
<th>bCity</th>
<th>propertyNo</th>
<th>pCity</th>
</tr>
</thead>
<tbody>
<tr>
<td>NULL</td>
<td>NULL</td>
<td>PA14</td>
<td>Aberdeen</td>
</tr>
<tr>
<td>B003</td>
<td>Glasgow</td>
<td>PG4</td>
<td>Glasgow</td>
</tr>
<tr>
<td>B002</td>
<td>London</td>
<td>PL94</td>
<td>London</td>
</tr>
</tbody>
</table>
Full Outer Join

- List branches and properties in same city and any unmatched branches or properties.

  SELECT b.*, p.*
  FROM Branch1 b FULL OUTER JOIN PropertyForRent1 p ON b.bCity = p.pCity;

- Includes rows that are unmatched in both tables.

- Unmatched columns are filled with NULLs.

<table>
<thead>
<tr>
<th>branchNo</th>
<th>bCity</th>
<th>propertyNo</th>
<th>pCity</th>
</tr>
</thead>
<tbody>
<tr>
<td>NULL</td>
<td>NULL</td>
<td>PA14</td>
<td>Aberdeen</td>
</tr>
<tr>
<td>B003</td>
<td>Glasgow</td>
<td>PG4</td>
<td>Glasgow</td>
</tr>
<tr>
<td>B004</td>
<td>Bristol</td>
<td>NULL</td>
<td>NULL</td>
</tr>
<tr>
<td>B002</td>
<td>London</td>
<td>PL94</td>
<td>London</td>
</tr>
</tbody>
</table>
Union, Intersection, Difference

- Used to combine results of two or more queries into a single result table.
  - Union of two tables, R and S, is table containing all rows in either R or S or both.
  - Intersection is table containing all rows common to both R and S.
  - Difference is table containing all rows in R that are not in S.
Union, Intersection, Difference

- All the tables must be *union compatible*.
  - Same number of columns
  - Same data type

- Operations
  - UNION (duplicates are omitted)
  - UNION ALL (duplicates are preserved)
  - MINUS (EXCEPT; Difference)
  - INTERSECT (Intersection)

- “ORDER BY” uses column numbers

- NULLS are ignored
Example: UNION

SELECT e.lname, e.fname, s.lname,
FROM employee e, employee s
WHERE e.superssn = s.ssn
UNION
SELECT e.lname, e.fname, 'NONE'
FROM employee e
WHERE e.superssn IS NULL
UNION
SELECT 'x', 'y', 'z'
FROM dual;
Next Session

- Nulls
- Constraints
- Views
- Transactions
- Access Control
- In-class exercises
Agenda

- More SQL Operators
- Unions, Intersections and Difference
- Constraints
- Views
- Transactions
- Access Control
- Nulls

PAY CAREFUL ATTENTION TO THE EXAMPLES!
OTHER SQL OPERATORS

IN

- `<column> IN <list>`
  - Thus, semantically equivalent to an ‘OR’ list
- Must examine EVERY value in the predicate
- Most implementations (like Oracle through 8i) permit only single column comparisons
- If `list` is empty (e.g., no rows returned), the result is FALSE.
- If `list` is a list of NULLs, the result will be UNKNOWN.
Exercise 1: Un-nested vs. Nested

- Review & execute the SQL statements in file Exercise1.sql
- Q1: Find the names of the sailors who have reserved boat number 103.
- Relational Algebra Expression:
  \[ \pi_{\text{sname}}(\sigma_{\text{bid}=103}(\text{Reserve} \bowtie \text{Sailor})) \]
- SQL Statements:
  - SELECT S.sname FROM Sailor S, Reserve R WHERE S.sid = R.sid AND R.bid=103;
  - SELECT S.sname FROM Sailor S WHERE S.sid IN (SELECT R.sid FROM Reserve R WHERE R.bid=103)
OTHER SQL OPERATORS

ANY

- Uses comparison operators: =, >, <, >=, <=, <>
- Takes a list as an argument
- x operator ALL (table_expression) evaluates to:
  TRUE if the expression
    x operator y
  evaluates to true for at least one value of y
  FALSE if the expression
    x operator y
  evaluates to false for all values of y (or if the list is empty)
  and evaluates to unknown otherwise
Example: ANY

- Find sailors whose rating is better than some sailor called Horatio.
- SQL Statement:
  - SELECT S.sid FROM Sailor S WHERE S.rating > ANY (SELECT S2.rating FROM Sailors S2 WHERE S2.sname = 'Horatio');
- S.rating > ANY ... S.rating is compared to each row that the subquery returns (equivalent to IN)
- Words to look for in the statement...“some”, “any”, etc.
OTHER SQL OPERATORS

- **ALL**
  - Uses comparison operators: =, >, <, >=, <=, <>
  - Takes a list as an argument
  - x operator ALL (table_expression) evaluates to:
    - TRUE if the expression
      \( x \text{ operator } y \)
    - evaluates to true for all values of y
    - FALSE if the expression
      \( x \text{ operator } y \)
    - evaluates to false for at least one value of y
    - and evaluates to unknown otherwise
Example: ALL

- Find sailors with the highest rating.
- SQL Statement:
  - SELECT S.sid FROM Sailor S WHERE S.rating >= ALL (SELECT S2.rating FROM Sailors S2);
- S.rating >= ALL ... S.rating is compared to each row that the subquery returns
- The outer WHERE clause is satisfied when S.rating >= each rating that the subquery returns
**EXIST**

- `<column> EXISTS <table subquery>`
- “Test for a non-empty set”
- In other words, the EXIST condition is used to test whether a specified table — usually a derived one — contains at least one row (i.e. to test whether the table in question is non-empty).
  - If set is non-empty, evaluates to **TRUE**
  - Otherwise, **FALSE**
- SELECT * option is better than an actual column (let optimizer select)
Example: EXISTS

- Find the names of sailors who have reserved boat number 103.
- SQL Statement:
  - SELECT S.sname FROM Sailor S WHERE EXISTS (SELECT * FROM Reserve R WHERE R.bid = 103 AND R.sid = S.sid);
- Do you know why this is an example of a correlated query?
OTHER SQL OPERATORS

- BETWEEN
  - y BETWEEN x AND z
  - Evaluates to $x \leq y$ and $y \leq z$
  - Inclusive
  - All (scalar) data types
  - If either $x$ or $z$ are *unknown* (NULL), evaluates to *unknown*
  - If both are *unknown*, evaluates to FALSE
OTHER SQL OPERATORS

- **LIKE**
  - Character strings
  - Inexact matches
    - _ = one and only one character
    - % = zero or more characters
  - Evaluates to *unknown* if either operand evaluates to NULL
Union, Intersection, Difference

- Used to combine results of two or more queries into a single result table.
  - Union of two tables, R and S, is table containing all rows in either R or S or both.
  - Intersection is table containing all rows common to both R and S.
  - Difference is table containing all rows in R that are not in S.
Union, Intersection, Difference

- All the tables must be *union compatible*.
  - Same number of columns
  - Same data type
- Operations
  - UNION (duplicates are omitted)
  - UNION ALL (duplicates are preserved)
  - MINUS (EXCEPT; Difference)
  - INTERSECT (Intersection)
- “ORDER BY” uses column numbers
- NULLS are ignored
Exercise 2: Regular vs. Union

- Q2: Find the names of sailors who have reserved a red or a green boat.
- Relational Algebra:
  - \( T_1 = \sigma_{(\text{color}='\text{red}') \lor (\text{color}='\text{green}')}(\text{Tempboats}) \)
  - \( \pi_{\text{sname}}(T_1 \bowtie \text{Reserve} \bowtie \text{Sailor}) \)
- SQL Statements:
UNION: Multiple Unions

SELECT e.lname, e.fname, s.lname,
FROM employee e, employee s
WHERE e.superssn = s.ssn
UNION
SELECT e.lname, e.fname, 'NONE'
FROM employee e
WHERE e.superssn IS NULL
UNION
SELECT 'x', 'y', 'z'
FROM dual;
Exercise 3:
Regular vs. Intersection vs. Subquery

Q3: Find the names of sailors who have reserved both a red boat and a green boat.

Regular SQL Statement:

```
```
Exercise 3: Regular vs. Intersection vs. Subquery

Q3: Find the names of sailors who have reserved both a red boat and a green boat.

Intersect SQL Statement:

```sql
SELECT S.sname FROM Sailor S, Reserve R1, Boat B1 WHERE S.sid = R1.sid AND R1.bid = B1.bid AND B1.color = 'red'
INTERSECT
SELECT S2.sname From Sailor S2, Reserve R2, Boat B2 WHERE S2.sid = R2.sid AND R2.bid = B2.bid AND B2.color='green';
```
Exercise 3:
Regular vs. Intersection vs. Subquery

- Q3: Find the names of sailors who have reserved both a red boat and a green boat.

- Subquery SQL Statement:

  ```sql
  ```
Exercise 4: Difference (MINUS)

Q4: Find the sids of all sailors who have reserved red boats but not green boats.

SQL Statement:

```
SELECT R1.sid FROM Reserve R1, Boat B1 WHERE R1.bid = B1.bid AND B1.color = 'red' MINUS
SELECT R2.sid From Reserve R2, Boat B2 WHERE R2.bid = B2.bid AND B2.color='green';
```
Constraints: Integrity Enhancement Features

Consider five types of integrity constraints:

- Required data
- Domain constraints
- Entity integrity
- Referential integrity
- Enterprise constraints
Integrity Enhancement Feature

Required Data - One Way

| position | VARCHAR(10) | NOT NULL |

Domain Constraints - Two Ways

(a) CHECK clause

| gender | CHAR | NOT NULL | CHECK (gender IN ('M', 'F')) |

(b) CREATE DOMAIN

CREATE DOMAIN DomainName [AS] dataType [DEFAULT defaultOption] [CHECK (searchCondition)]

For example:

CREATE DOMAIN GenderType AS CHAR

CHECK (VALUE IN ('M', 'F'));

Usage

| gender | GenderType | NOT NULL |
Check constraint

- Makes sure that values meet certain conditions
- Expands on database’s ability to enforce business rules
- Must be true for every row in the table
- May not query other rows in the table or other tables
- Example
  - CREATE TABLE WORKER(
  -   NAME CHAR(25) PRIMARY KEY,
  -   AGE NUMBER CHECK(AGE BETWEEN 18 AND 65),
  -   LODGING CHAR(15));
Check constraint

- You can't refer to pseudo-columns (SYSDATE, UID, USER, USERENV, CURRVAL, NEXTVAL, ROWNUM)
- Constraints can be named, otherwise Oracle will generate one
  - SYS_C####
- Constraints may be disabled and enabled
- Multiple columns may be referred to in a constraint
  - CONSTRAINT EMP_KEY PRIMARY KEY (SSN, AGE)
Entity Integrity

- Enforced using Primary Key
  - Primary key of a table must contain a unique, non-null value for each row.
  - Only one Primary Key is allowed per entity/table
  - Multiple Alternate keys are allowed. These may or may not be unique

    PRIMARY KEY(staffNo)
    PRIMARY KEY(clientNo, propertyNo)
    UNIQUE(telNo)
Referential Integrity

- Data Integrity between 2 or more Tables
  - Is Enforced with Foreign Keys
  - Referential integrity means that, if FK contains a value, that value must refer to existing row in parent table.
  - Foreign Key is column or set of columns that links each row in child table to a row of parent table via matching keys, FK in child table to the PK of the parent table.
  - ISO standard supports definition of FKS with FOREIGN KEY clause in CREATE and ALTER TABLE:

    FOREIGN KEY(branchNo) REFERENCES Branch
Referential Integrity

- RI effects on DML statements
  - Any INSERT/UPDATE that attempts to create FK value in child table without matching candidate key value in parent is rejected.
  - Action taken that attempts to update/delete a candidate key value in parent table with matching rows in child is dependent on referential action specified using ON UPDATE and ON DELETE subclauses (of child table)
    - CASCADE
    - SET DEFAULT
    - SET NULL
    - NO ACTION

FOREIGN KEY (staffNo) REFERENCES Staff  ON DELETE SET NULL
FOREIGN KEY (ownerNo) REFERENCES Owner ON UPDATE CASCADE
Enterprise Constraints

- Enforcing Business Rules of the Enterprise
  - Several methods available
  - Could use CHECK/UNIQUE in CREATE and ALTER TABLE.
  - Also have:
    - CREATE ASSERTION AssertionName CHECK (searchCondition)

For Example
CREATE ASSERTION StaffNotHandlingTooMuch
CHECK (NOT EXISTS (SELECT staffNo
  FROM PropertyForRent
  GROUP BY staffNo
  HAVING COUNT(*) > 100))
Exercise 5: Constraints

- Re-write the SQL statements from Exercise1.sql to include the following constraints:
  - Primary key in Sailor is sid
  - sname in Sailor cannot be null
  - Primary key in Boat is bid
  - The combination of bname & color must be unique
  - sid in Reserve references sid in Sailor
  - bid in Reserve references bid in Boat
Views

- Virtual relation that does not actually exist
- Contents of a view are defined as a query on one or more base relations.
- With **view resolution**, any operations on view are automatically translated into operations on relations from which it is derived.
- With **view materialization**, the view is stored as a temporary table, which is maintained as the underlying base tables are updated.
Views

- DROP VIEW ViewName [RESTRICT | CASCADE]

- CREATE VIEW ViewName [(newColumnName [, ...]) ]
  AS subselect
  [WITH [CASCADED | LOCAL] CHECK OPTION]

- The *subselect* is known as the defining query.
- WITH CHECK OPTION
  - Ensures that if a row fails to satisfy WHERE clause of defining query, it is not added to underlying base table.
- Need SELECT privilege on all tables referenced in subselect and USAGE privilege on any domains used in referenced columns.
View Resolution

Defining a view –
CREATE VIEW StaffPropCnt (branchNo, staffNo, cnt)
    AS SELECT s.branchNo, s.staffNo, COUNT(*)
        FROM Staff s, PropertyForRent p
        WHERE s.staffNo = p.staffNo
        GROUP BY s.branchNo, s.staffNo;

Using a View -
SELECT staffNo, cnt
    FROM StaffPropCnt
    WHERE branchNo = 'B003'
    ORDER BY staffNo;
Views - Restrictions

Several restrictions exist on creation and use of views.

(a) Aggregate Function based Column:

- That column may appear only in SELECT and ORDER BY clauses of queries that use the view.
- That Column CANNOT be used in WHERE clause, NOR as an operand in an aggregate function in the query based on view.
  - For example, following query would fail:
  
  ```sql
  SELECT COUNT(cnt)
  FROM StaffPropCnt;
  ```
  
  Similarly, following query would also fail:
  ```sql
  SELECT *
  FROM StaffPropCnt
  WHERE cnt > 2;
  ```
Views - Restrictions

(b) Grouped view may never be joined with a base table or a view.

- For example, StaffPropCnt view is a grouped view, so any attempt to join this view with another table or view fails.
Views - Updatability

- Views cannot be used to update the base tables
- A view is updatable if and only if:
  - DISTINCT is not specified.
  - Every element in SELECT list of defining query is a column name and no column appears more than once.
  - FROM clause specifies only one table, excluding any views based on a join, union, intersection or difference.
  - No nested SELECT referencing outer table.
  - No GROUP BY or HAVING clause.
  - Also, every row added through view must not violate integrity constraints of base table.
Transactions

- SQL defines transaction model based on
  - COMMIT and ROLLBACK.

- Transaction
  - A logical unit of work with one or more SQL statements that is guaranteed to be atomic with respect to recovery.
  - Begins with a transaction-initiating SQL statement (e.g., SELECT, INSERT).
  - Changes made by transaction are not visible to other concurrently executing transactions until transaction completes.

- Transactions can complete in one of four ways:
  - COMMIT: Successful. Make changes permanent.
  - ROLLBACK: Abort transaction. Back out all changes.
  - Embedded SQL
    - Implied COMMIT: Successful program end.
    - Implied ROLLBACK: Abnormal Program end
Access Control

- Determines access privileges on the DB objects

- Owner of table must grant other users the necessary privileges using GRANT statement.
CREATE USER

Syntax

CREATE USER *username* IDENTIFIED {by *password* | externally}

Examples

a) Create user Dora with and assign “psyche”

CREATE USER Dora IDENTIFIED BY psyche;

b) Change Dora’s password to “avocado”

ALTER USER Dora IDENTIFIED BY avocado;
CREATE USER

- Reserved Userids and Passwords

<table>
<thead>
<tr>
<th>USERID</th>
<th>PASSWORD</th>
</tr>
</thead>
<tbody>
<tr>
<td>system</td>
<td>manager</td>
</tr>
<tr>
<td>sys</td>
<td>change_on_install</td>
</tr>
<tr>
<td>internal</td>
<td>oracle</td>
</tr>
<tr>
<td>scott</td>
<td>tiger</td>
</tr>
</tbody>
</table>

Do NOT use the above reserved user ids when creating users!
GRANT

Syntax

```
GRANT {PrivilegeList | ALL PRIVILEGES}
ON ObjectName
TO {AuthorizationIdList | PUBLIC}
[WITH GRANT OPTION]
```

Examples

a) Give Manager full privileges to Staff table.

```
GRANT ALL PRIVILEGES
ON Staff
TO Manager WITH GRANT OPTION;
```

b) Give SELECT and UPDATE on column salary of Staff.

```
GRANT SELECT, UPDATE (salary)
ON Staff
TO Personnel, Director;
```
REVOKE

- REVOKE takes away privileges granted with GRANT.

  REVOKE [GRANT OPTION FOR]
  {PrivilegeList | ALL PRIVILEGES}
  ON ObjectName
  FROM {AuthorizationIdList | PUBLIC}
  [RESTRICT | CASCADE]

- GRANT OPTION FOR
  - Allows privileges passed on via WITH GRANT OPTION of GRANT to be revoked separately from the privileges themselves.

- REVOKE fails if it results in an abandoned object, such as a view, unless the CASCADE keyword has been specified.
REVOKE

- Example
  
  Cancel SELECT & UPDATE privileges for Bob and Mary

  REVOKE SELECT, UPDATE ON employee, project FROM bob, mary;
NULLs

Meanings:
- Not Applicable
- Unknown

NULLs propagate in math expressions and in functions

select NVL(salary*1.5, -99) from employee where lname = 'x';

Converting NULLS
- NVL(column, value) [COALESCE(column, value)]
NULLs

- NULLs are permitted by default
- SUM, MIN, MAX, AVG, COUNT, DISTINCT all exclude NULLS
- COUNT includes NULLs
- NULLs do not participate in comparisons
- NULLs are not included in an INDEX

BOTTOM LINE: NULLs cannot be used in calculations
Exercise 5: NULLS

- Find all employees that work in the SALES department and earn over 1000.

- Review all of the Exercise5*.sql files
Next Session

- Introduction to PL/SQL
- Handout Project Proposals
- Project Team Sign-up