SQL Notes

SQL (Structured Query Language)

DAO Design Pattern: Encapsulates and abstracts data accessed behind an interface that we provide our business logic layer with.

Applications often use a factory to decide between implementations at runtime.

JDBC (Java Database Connectivity): How we connect Java to SQL; use the java.sql package and ojdbc.jar

Important Interfaces

Connection: Manages the connection to (session with) the database. Allows us to execute SQL queries and return results. Has information about DB tables, stored procedures, and other details (use the getMetaData method).

Statement: takes an SQL statement as a string, executes, and returns the result. VERY BAD because it is vulnerable to SQL injection.

PerparedStatement: Executes a pre-compiled SQL query; excellent for queries executed multiple times.

CallableStatement: Extends the PreparedStatement; allows us to execute stored procedures.

ResultSet: Table of data representing a DB result set. Generated by executing a query passed in executeQuery(). Maintains a cursor pointing to the current row when iterating through the set by calling the next() method (cursor is initially positioned above the first row). The next() method moves the cursor to the next row and returns false when there are no more rows.

PL/SQL (Procedural Language extension of SQL – Oracle SQL's Procedural Language)

Sequences: Good for PK management

```
CREATE SEQUENCE [name]

MINVALUE [val]

MAXVALUE[val]

START WITH [val]

INCREMENT BY [val]

CACHE[amount]*
```

* How many values will be stored in memory for faster access; this portion is optional

Triggers: Block of code that is automatically executed when the specified event happens (before/after inserts, updates, deletes, etc.).

```
CREATE [OR REPLACE] TRIGGER [name]

BEFORE [INSERT | UPDATE | DELETE] ON [table]

BEFORE EACH ROW*

BEGIN

[code]

END;
```

* Required to see and manipulate row data

Indexes: Apply an index to a column to enforce a physical in-memory ordering of its rows. Speeds up searching, slows down inserting/updating PKs, FKs, and unique columns, creating "backing" indexes – multiplicity generated index on the specified columns (cannot add additional indexes to these columns).

```
CREATE INDEX [name] ON [table] [table column] ASC | DESC;
```

Views: Virtual table returned by query.

Cursors: Oracle creates a "context area" with all information necessary for processing an SQL statement. A <u>cursor</u> is a pointer to that context area. The "active set" is the row(s) returned by a statement and is held by the cursor.

Implicit Cursor: Created by Oracle whenever we execute a DML statement; we have no control over these.

Explicit Cursor: Programmer defined for obtaining more control over context area; declare cursor to initialize memory, open cursor to allocate memory, fetch cursor to retrieve data, and close cursor to release memory.

Functions: Blocks of code we can execute.

• Can take 0 or many parameters.

• Full DML (and TCL capabilities)

```
CREATE [OR REPLACE] PROCEDURE [name]

[list of parameters]

IS | AS

[declarations]
```

```
BEGIN

[code]

EXCEPTION

[exception handling code]

END;
```

Transactions: Units of work done on a database; may include many DML operations.

TCL: Transaction Control Language

• COMMIT – Persist all changes

• ROLLBACK – Discard changes since last commit

• SET SAVEPOINT – Creates a point during a TX between commits to roll back to

Properties of a Transaction (A.C.I.D.)

Atomicity: "All or nothing"; either all operations in a transaction execute successfully, or no commit is made.

Consistency: DB is in a valid state (according to existing structure and constraints) after commit.

Isolation: The system state during concurrent transactions is the same as if the transactions were sequential.

Durability: All commits are final and cannot be rolled back, even in the case of a system failure.

For isolation, there are 3 levels to which your DB can isolate its transactions (TX). These are called TX isolation levels. Each prevents your TX from a different set of problems.

X = Problem can occur

✓ = Problem is prevented

Isolation Level	**Dirty Read**	**Non-Repeatable Read**	**Phantom Read**
Read Uncommitted	X	X	X
Read Committed	✓	X	X
Repeatable Read	✓	✓	X
Serializable	✓	✓	✓

In summary, TX isolations levels are a measure of the extent to which TX isolation succeeds. They are defined by the presence/absence of the following phenomena:

Dirty Read: Occurs when a TX reads data that has not yet been committed.

TX_1	TX_2
Updates row A	----
----	Reads row A
Commits	----

Leaves open possibility for TX_1 to roll back changes before commit, so TX_2 has read data that is considered never to have existed.

Non-Repeatable Read: Occurs when a TX reads the same row twice, but gets different data each time.

TX_1	TX_2
Reads row A	----
----	Updates/Deletes row A
----	Commits
Re-reads row A	----

Phantom: A row that matches the search criteria but is not initially seen.

TX_1	TX_2
Reads rows that match criteria A	----
----	Inserts/Updates row to match criteria A
Re-executes query A, sees different set of rows	----

Read Uncommitted: TX are not isolated. TX running at this level are usually read-only.

Read-Committed: TX waits until rows write-locked by other transactions are unlocked. This prevents it from reading "dirty" data, aka "dirty reads". Also holds a read lock (if it only reads the row) or write lock (if it only updates/deletes the row) on the current row to prevent other TX from affecting it. The TX releases read locks when it moves off of the current row, and releases write locks when it is committed or rolled back.

Repeatable Read: Holds read lock on all rows it returns to the app and write locks on all rows it updates/deletes.

Serializable: TX holds a read lock (if it only reads) or write lock (if it updates/deletes) on the range of rows it affects.